学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权江苏大学可以将本学位论文的全部内容或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保　密　□，在　　年解密后适用本授权书。

本学位论文属于

不保密　☑。

学位论文作者签名：黄杰　　指导教师签名：

2010年12月19日　　2010年12月19日

独创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已注明引用的内容以外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：周杰

日期：2010 年 12月 19 日

摘　要

随着信息化时代的到来，计算机网络技术，图像处理技术与通信技术飞速发展，极大的推动了监测技术的不断发展与更新。面对内河海上运输的迅速发展，海事监测将是保障船舶航行安全的重要手段。海事视频监测相比传统的有线监测有很多优点，已广泛应用于社会生活的各个领域。为了满足视频监测在海事监控中的应用需要，论文研究和设计了无线视频监测系统，对视频监测的采集、编码、传输等模块进行了研究。

通过对嵌入式技术、数字图像技术及无线网络传输等技术的深入研究，提出了基于 S3C6410 平台的无线视频监测系统的总体方案。指出了采用 S3C6410 处理器设计的视频处理的优势。重点对监测终端部分进行研究设计：硬件平台根据视频数据采集、编码及传输需要选择了 ZC0301 摄像头、基于 S3C6410 处理器的嵌入式开发板以及 3G 模块；软件平台采用嵌入式 Linux 操作系统。论文依据硬件平台和嵌入式 Linux 构建了无线视频监测终端的软硬件框架，并进行开发环境的构建，包括：建立嵌入式系统的交叉编译开发环境，对 Bootloader 的启动过程分析，实现 Bootloader 的移植；针对 S3c6410 目标板，完成 ARM-Linux 操作系统的内核配置和移植。根据应用软件的总体功能，对视频采集编码模块和视频传输模块进行了设计：其中，深入分析 H.264 的编码结构，对其中复杂度较高的运动估计算法和帧内预测算法进行改进和优化，提高编码效率，使编码器达到满足嵌入式应用的需要；融合采集和编码程序并对移植后的编码器针对 S3C6410 平台进行优化完成了视频的采集压缩模块设计。提出基于 RTP 的传输控制策略，通过 socket 套接字进行无线视频数据收发，完成了对视频传输模块的设计。最后，构建测试环境，对编码器性能和监控效果进行测试分析。

测试结果表明，在软硬件环境下，所设计的监测系统可以对 QCIF 分辨率视频进行采集并进行 H.264 编码压缩，经优化的编码器编码速度可以达到 24fps，基本可达到实时的效果，还可以对 S3C6410 平台进一步优化和采用更高性能的处理器等手段。

关键词：海事　S3C6410　H.264　Linux　监测系统　RTP　3G

ABSTRACT

With the coming of information age, computer network technology and image processing technology and the rapid development of communication technology greatly continuously promoted the development of monitoring technology. Marine video surveillance which has many advantages compared with the traditional wired one, had been widely used in various fields of social life. To meet the application needs of wireless video surveillance cameras in the helmets of the police, this paper study and design embedded wireless video monitoring system. It in-depth studies the video surveillance capture, coding and transmission.

Through in-depth study on embedded technology and digital imaging technology and wireless network transmission technology, a precept of wireless video surveillance system design program based on S3C6410 is presented.This issue focuses on the the research and design of the monitoring terminal.In terms of hardware, select the ZC0301 camera, embedded development board which based on S3C6410 processor and 3G modem according to the needs of video surveillance capture，coding and transmission. The embedded Linux operating system is selected as software platform. According to the hardware platform and embedded Linux,the thesis build a wireless video surveillance hardware and software framework for the terminal, and build development environment, for example, building the cross-compiler development environment based on the embedded systems，analysing the Bootloader start process，the Bootloader transplantation，ARM-Linux kernel configuration and migration for S3C6410 target board. The paper designes the video capture and transmission modules according to application software: in-depth analysis the H.264 main modules, on which the higher complexity of motion estimation and intra prediction algorithm is improved and optimized to improve the coding efficiency, so that the encoder meet the needs of embedded applications, through the integration of collection and coding procedures and transplantation encoder and optimized for the S3C6410 platform to complete video capture and compression module design; to study the transmission control method based on RTP, to send and receive wireless video data by socket to completed the design of video transmission module; Finally, building test environment to analysis results of the encoder performance.

The test results showed that in the hardware and software environment, the design of the control terminal of the QCIF resolution video capture and encoding according to H.264 standard, the optimized encoder speed can reach 24 per second, the basic quasi-real-time results can be achieved. To achieve real-time video surveillance, it also depends on further optimization for the Xscale platform and uses higher performance processors and other means.

Key Words: Marine S3C6410 H.264 Linux Monitoring System RTP 3G

目　录

第一章 绪论

1.1 研究背景

随着生活水平的提高，人们对发生事件现场的状况提出了更高的要求。我国船舶数量的不断增加，船舶的管理，船舶遇险后的发现、搜救指挥愈发显得重要。传统的船岸通信不能完全适应通信发展上数字化和宽带化的需要，特别是不能很好满足船岸应急通信中图象传输的需求；此时，依靠无线传输技术的视频监测系统的优势就体现出来了。

在技术方面，嵌入式技术、视频编解码技术和无线传输技术的快速发展，使得嵌入式无线视频监测成为可能。随着视频编解码算法的改进，在同等清晰度下，压缩比进一步提高，传输的数据量减小。而无线传输技术的发展更是立竿见影，3G 时代的到来，给予无线视频传输更充足的带宽。这些为实现嵌入式视频监测提供了良好的平台。

嵌入式视频监测相比其他监测系统，有如下优点：

1) 嵌入式视频监测与网络结合，直接接入网络，即插即用，扩大了监测地域，增加的设备只是 IP 地址的扩充，使用方便，节省成本。

2) 嵌入式视频监测系统多采用嵌入式实时多任务操作系统，系统的实时性、稳定性、可靠性大大提高，往往无需专人管理，可用于很多特殊情况使用。

3) 嵌入式视频监测通过联网可以使监测更加灵活，随时随地得到现场情况，由于网络的双向传输性，监测中心还可以通过控制监测终端来实现全方位的监测；通过网络对现场进行控制，还可以进行图像分析辨识，实现对监测现场异常情况的实时监测。

嵌入式视频监测的国内外研究和应用现况如下：由于社会、经济等方面的因素，我国的监测领域起步比较晚，在 90 年代以后才得到了较快发展。目前，我国拥有很多专门从事视频监测生产与研究的企业，如海康威视、天津三星等国内知名企业；许多大专院校和科研机构也都从事这一领域的研究。由于国外发达国家监测技术比我国起步早很多年，掌握着很多核心技术，目前已经形成了不少知名品牌，如：索尼、博世、松下等。随着监测领域研究的的深入，其相关的应用也越来越广泛，如将其应用于煤矿的生产中，减少生产事故的发生[1]；在远程医疗领域采用监测技术为偏远落后地区提供医疗服务[2]。

1.2 海事视频监测的重要性及其特点

随着港口建设的发展，经济对运河、海运的依赖日益提高，视频监测系统将逐渐成为各大港口实现安全生产，减轻现场工作人员的劳动量和劳动强度，及时发现各种危险状况，制止事故发生的主要工具。

与其他的监控相比，海事监控具有以下特点：

（1） 需要监控的范围广，包括海面（江面）、停泊区、港口、码头、货场。保税区等等，要将这些地区完整地监控起来，必须解决现场采集、远程管理等问题。

（2） 制定内河交通事故等级标准和对应的内河搜救应急措施，开发内河交通事故专家系统，对交通事故自动进行登记划分并决策选择不同应急措施策略。

（3） 采用模式识别和图像处理技术，对运行的船舶状态使用摄像头实时监测跟踪，对异常状态自动预警。

（4） 采用3G网络，联动控制中心与其他公共服务中心联网，综合各种应急服务资源，用于报告紧急事件和紧急联动，并统一指挥，联合行动。

（5） 采用数据挖掘技术，对船舶运输状态分析，预测可能出现的异常状态，提前采取应急措施。

1.3 本文研究内容

课题的目的是研究和设计无线海事视频监测终端，该终端的功能是将采集到的视频信息进行H.264编码，将压缩后的视频数据流通过3G无线网络传输到监测服务器中，同时监测终端完成压缩视频流的存储备份。

所研究的嵌入式无线视频监测终端顺应监测领域的发展趋势，结合先进的H.264视频压缩技术和网络传输协议，并将近几年来，已经成熟的无线通讯技术运用到视频监测中，具有一定的理论和实践价值。该视频监测终端可应用于海事监控中[3]，能够真正实现对船舶的实时监控，在发生意外的情况下，不需要船舶上的人员参与，指挥人员在指挥中心可以及时看到现场的实际情况，能够迅速做出指挥判断。同时监测终端具有低成本、小型化、扩展性强等优点，也可扩展到其他的无线多媒体应用中。

论文以海事监控为应用对象，通过深入分析H.264/AVC[4]视频编码技术和嵌入式技术，将采集到的视频进行H.264视频编码压缩，同时实现基于RTP的H.264传输控制方法，完成了嵌入式无线监测终端的设计。全文组织结构如下：

第一章：简单阐述了课题的研究背景及监测系统研究应用现状，引出课题的研究目的和意义。

第二章：提出嵌入式无线视频监测系统的基本框架，对监测系统分别从硬件ARM11处理器S3C6410、3G模块和软件框架两方面阐述设计的思路。

第三章：阐述了嵌入式Linux[5]的发展与特点，建立交叉编译开发环境，针对S3C6410

处理器分析 Bootloader 的启动过程，对其进行设计移植，对内核进行了相关配置、修改与编译，并对其进行了移植，同时完成 Linux 根文件系统的设计和移植，实现嵌入式系统软件的开发与移植。

第四章：深入分析了 H.264 视频编码结构，对其中复杂度较高的运动估计算法和帧内预测算法进行了改进与优化，提高了编码效率，为下一步编码移植做准备。

第五章：通过 V4L2 编程完成了视频采集，将视频采集程序和编码算法进行融合，对 H.264 编码器移植到 S3C6410 平台上，并针对该平台进行优化，完成视频的采集压缩模块设计。

第六章：提出基于 RTP 的 H.264 传输控制方法，重点分析了基于 JRTPLIB 库的传输控制编程，并给出其实现步骤。

第七章：结合 3G 无线网络的接入方法和原理，对整个系统进行测试，并进行相应的结果分析。

第八章：总结课题的工作及特点，对下一步工作进行展望。

第二章 海事视频监测系统总体方案

海事视频监测是在软硬件协同工作的基础上运作的，良好的总体方案是视频监测系统能够开发成功的关键。本章提出无线视频监测系统的基本框架，重点对监测终端从硬件、软件两个方面分别介绍。

2.1 海事视频监测系统总体方案

本课题研究的海事视频监控系统，基于嵌入式系统的 RISC 通用处理器，采用免费开源的嵌入式 Linux 操作系统，在功能上实现视频图像的采集、压缩、存储与无线收发。它采用先进的 H.264/AVC 视频压缩标准对原始视频数据进行压缩处理，并保存在嵌入式系统中，同时对压缩后的视频进行无线发送。

根据上述设计思想，提出该系统的总体设计框架如图 2.1 所示：

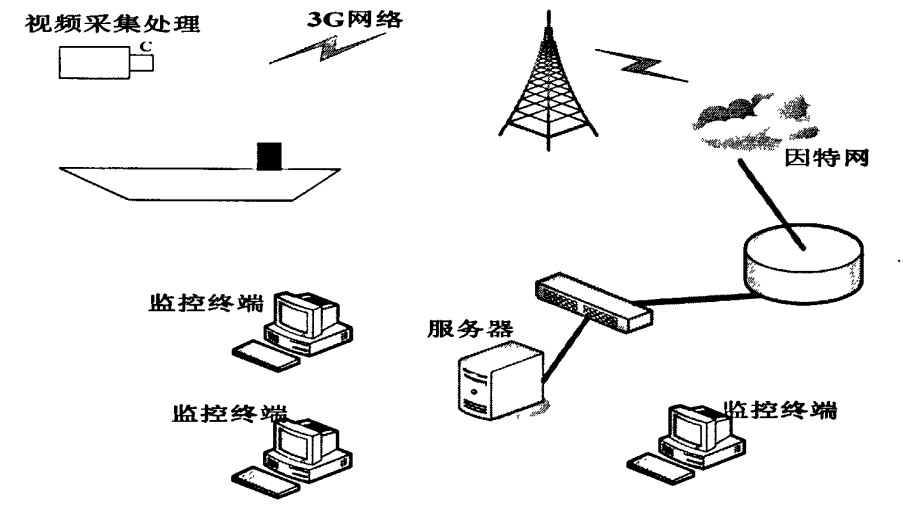


图2.1 海事视频监测系统基本框架

整个海事视频监控系统分为软件和硬件两大部分。在硬件部分，以 RISC 通用处理器为核心，由 SDRAM、FLASH 等存储器件和各种通信接口构成嵌入式硬件平台。通过 USB HOST 接口接入摄像头外设，以捕获视频原始数据，实验构成中将采集压缩后的视频存入 SD 卡，并通过 3G 无线收发模块将压缩过后的视频流打包进行无线传输到本地 PC 上。在软件部分，引导程序与嵌入式 Linux 内核、必要的设备驱动程序、根文件系统一起形成嵌入式系统的基本运行环境，对应用层的用户应用程序作必要的支持。在本文中应用程序是视频采集压缩程序与 socket 收发程序。

论文重点对系统的监测系统部分进行研究和设计，下面分别对监测的软硬件部分进

行说明。

2.2 S3C6410 视频编解码功能特点

S3C6410 是基于 16/32bit RISC[6] 内核的高性能处理器解决方案，用于移动电话和通用应用。为了给 3G 业务提供最佳的硬件性能，S3C6410 采用 64/32bit 内部总线架构，内部集成了多个功能强大的硬件加速器，如移动图像处理、显示控制和图像缩放。集成多格式编解码器（MFC）视频会议及 NTSC 和 PAL 格式的 TV 输出。此外，S3C6410 包含高级 3D 图形加速器，三角形生产率 4M/s ,带 OpenGLES1.1/2.0，D3DM API 接口。硬件结构图如下图 2.2 所示，其中，ARM 控制器和 3G 模块是监测系统重要组成部分。

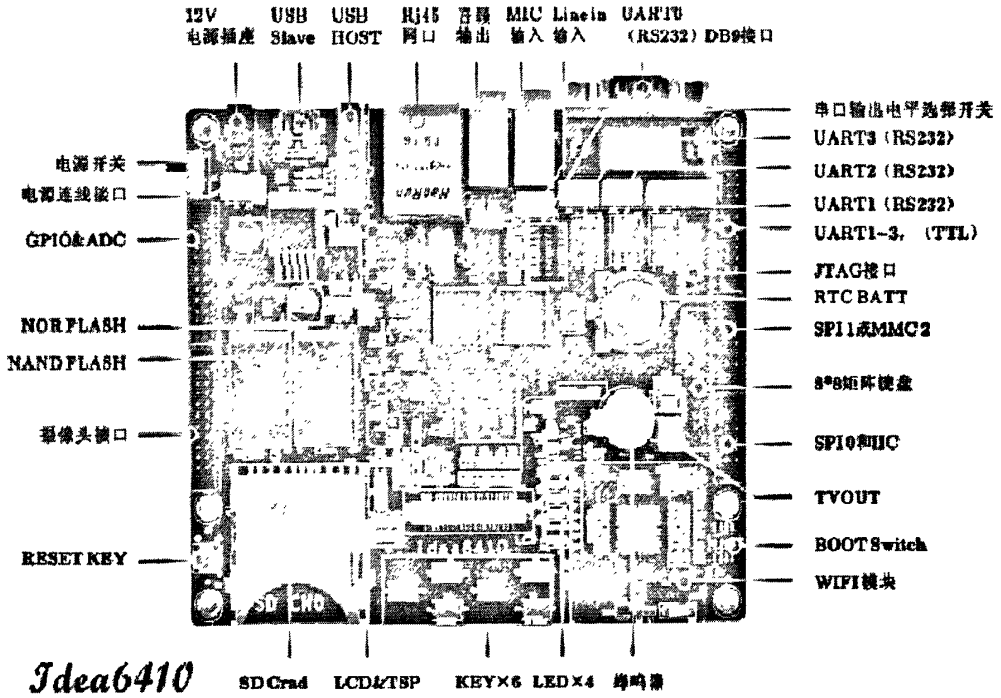


图2.2 监测终端硬件结构框图

2.2.1 S3C6410 处理器简介

S3C6410 是韩国三星电子基于 ARM1176JZF-S 内核构建的高性能多媒体应用处理器，它不仅具有强大的硬件编解码单元，完善的外设，而且拥有高达 667MHz 的运行频率。包括分立的 16KB 指令和 16KB 数据 Cache，16KB 指令和数据 TCM。还包括一个完全的 MMU 来处理虚拟存储管理。ARM1176JZF-S 是一款单芯片 MCU，支持 JAVA 加速，有一个专用的矢量浮点协处理器能高效执行各种加密运算及高品质 3D 图形应用。

S3C6410 有极佳的外部存储器接口能力，可以满足高端通信业务的带宽要求。存储器系统有双总线架构，一路用于内存总线、一路用于 Flash 总线，可以并行访问。DRAM

端口能配置为移动 DDR 或标准 SDRAM。Flash/ROM 端口支持 NANDFLASH、NORFLASH、OneNANn、CF 和 ROM 类型外部存储器。为消减系统总成本和增强总体功能，S3C6410 集成了许多硬件外设，如 Camera 接口、TFT-24bit 真彩色 LCD 控制器、电源等系统管理、4 通道 UART、32 通道 DMA、4 通道定时器、通用 I/O 端口、IIS、IIC 总线接口、USBHost、USBOTG(480Mbps)、3 通道 SD/MMC Host 控制器及时钟生成 PLL。采用 90nmCOMS 工艺，低功耗、简洁、精美且全静态设计使得 S3C6410 非常适合对成本、功耗敏感的应用。

2.2.2 S3C6410 视频编解码功能

S3C6410 编解码单元包括 BIT 处理器和视频编解码模块。BIT 处理器能形成比特流和控制视频编解码，为了加速比特流的处理，BIT 处理器内嵌了一些硬件加速器。在 BIT 处理器上运行的程序和数据通过 APB 和 AXI 总线下载。编解码单元在硬件上经过优化减少了逻辑门的数量并且多格式视频编解码共享大部分子模块。运动估计模块使用了一块查找 RAM 来减少外部 SDRAM 的带宽。一般来说，运动估计模块能多次读取参考像素值，并且把参考像素值从外部 SDRAM 装载到查找 RAM 并存储。查找 RAM 的存取是通过 AHB 总线。宏分析模块通过安排视频编解码功能模块的处理流来减少装载到 BIT 处理器中复杂的固件程序。编解码单元还包含了旋转和镜像模块，如果压缩处于旋转和镜像源图像时，BIT 处理器不需要额外的带宽。但是，在解压过程中，旋转和镜像会把解压图像写到外部存储器中。主处理器与编解码单元的通信是通过 API 接口，为了使视频编解码更便利和具有容错能力，与比特流相关的处理都由 BIT 处理器来处理。

编解码单元能处理 MPEG4、H.263、H.264 编解码和 VC-1 解码，它还能同时处理图像的压缩和解压，也能同时对不同格式的图像进行处理，例如，对 MPEG4 压缩的同时还能完成 H.264 的解压。BIT 处理器能处理比特流和控制视频编解码硬件，一般目的寄存器和中断寄存器可用于主处理器和编解码单元通信。BIT 处理器通过 APB 总线接口与主处理器通信，通过 AXI 总线接口与外部存储器通信。

2.3 3G 网络和 3G 模块

2.3.1 3G 网络特点

由于采用了分集接收、信道均衡、功率控制等技术，使得 3G 系统的无线空中接口技术相比 2G/2.5G 系统有了很大的提高，在运动条件下能够给用户提供 384kbps 的信道带宽，在静止条件下提供的带宽可以达到 2Mbps。

根据无线空中接口技术的不同，现有 3G 技术大致可以分为 3 类:由欧洲和日本提出的 WCDMA 系统、由美国提出的 CDMA2000 系统和由中国提出的 TD-SCDMA 系统。

其中 WCDMA-FDD 系统采用码分多址、频分双工的工作方式，上下行频率间隔为95MHz，采用直接序列(DS)作为信息扩频方式。而 WCDMA-TDD 系统和 TD-SCDMA 系统都采用码分多址/时分多址/时分双工的工作方式，两者的区别在于扩频码的速率不同，前者采用高码片速率 3. 84Mcps，而后者采用地码片速率 1. 28Mcps。CDMA2000 采用码分多址、频分双工的工作方式，在下行链路传输中，定义了直扩和多载波两种工作方式，码片速率分别为 3. 6864 Mcps 和 1. 22Mcps。

现有支持 IMT-2000 的标准化组织主要是 3GPP 和 3GPP2。其中 3GPP 称为 3G 合作伙伴项目，负责制定以原有 GSM 核心网络演变的 3G 标准，以 WCDMA、TD-SCDMA 为空中接口技术。3GPP 目前已经存在多个版本，每一个版本都有不同之处，目前最成熟的是 R5 版，正在制定的是 R6 版。3GPP 称为 3G 合作伙伴项目 2，负责制定以 ANSI-41 核心网为基础，CDMA2000 为无线空中接口的 3G 技术规范。现在成熟的 3GPP2 版本主要有 CDMA2000-1X 和 CDMA2000-1XEV，其中后者是前者的增强技术

2.3.2 3G 模块

采集终端与 3G 通信网络[7]之间的连接是通过 3G 模块终端来实现的。目前市面上已经有许多厂家生产的多种 3G 模块，如西门子、华为、中兴等。本系统采用的是 UT_TD-SCDMA 3G 模块，该模块的特点是稳定性好。

UT_TD-SCDMA 3G 模块支持 CDMA 800M/1900MHz 频段，内置双天线，支持 CDMA EVDO Rev A，支持扩展外部存储器。其 TD-SCDMA 工作频段为 2010-2025MHz。支持多通道复用协议，集成 TCP/IP 协议。支持 AT 指令控制模块实现数据的传输。

AT 指令的概述：

AT 指令是 Hayes 公司为 MODEM(调制解调器)制定的一个控制指令集，目前在工业界已变成了一个事实标准，广泛应用于对 Modem 和 MS(移动台)的控制。

所有 AT 指令都以“AT”开头，以<CR>结尾，不区分大小写。主要有四种格式：

1) 无参数指令：指示模块做什么，模块根据内部参数完成命令，并应答；

2) 查询指令：查询该指令当前设置的值，模块返回设置值；

3) 帮助指令：用来列出该指令的可能参数，模块返回列表；

4) 带参数指令： 设置模块的相应参数。

AT 指令的返回值主要有以下两种情况：

1) 指令的操作结果报告：返回最近一条 AT 指令的操作结果，若指令操作错误，则返回错误代码。

2) 事件报告：当网络有下发事件时，如收到短信、来电振铃等，模块会主动将事件报告给客户。

课题所使用的UT_TD-SCDMA 3G模块如下图2.3所示：

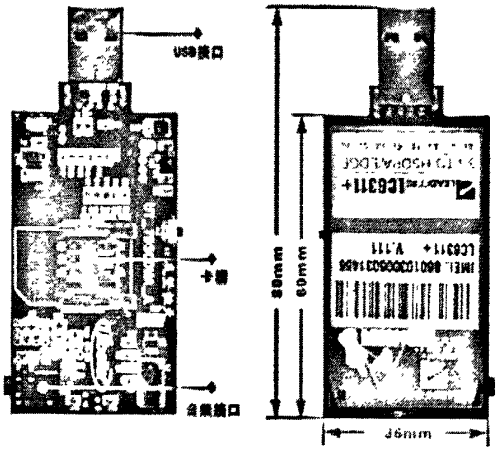


图2.3 3G模块

2.4 视频监测终端软件框架

监测终端软件框架主要包括：嵌入式Linux系统的构建及应用软件的开发。构建嵌入式系统软件包括：交叉编译开发环境的建立、Bootloader 的设计移植、Linux 内核的编译和移植、根文件系统的构建。该部分的工作主要为应用软件搭建一个系统平台。应用软件包括视频采集编码模块设计和视频传输模块设计。视频采集压缩模块完成对视频信号的采集，将USB摄像头采集的信息以文件的形式保存或者放入内存缓冲区，由X264视频压缩编码视频缓冲区数据，对其进行压缩处理后形成.264文件，用SD卡进行存储，另一方面将压缩后的视频流通过视频传输模块传输到监测服务器。其软件结构如下图2.4所示：

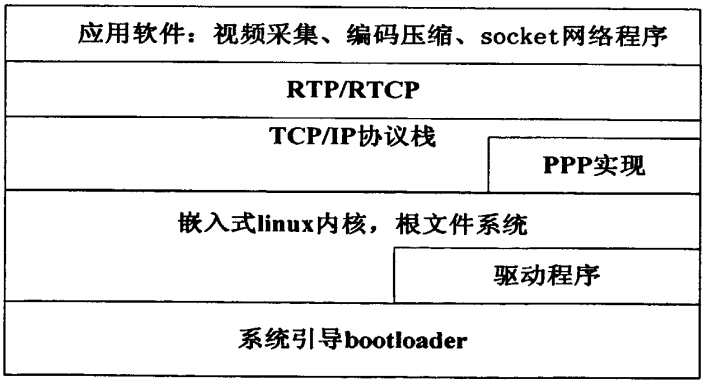


图2.4 监测终端软件结构图

2.4.1 嵌入式系统软件的构建

1) 嵌入式交叉开发环境的建立

嵌入式开发的目标平台确定后，首先要建立交叉开发环境。交叉开发环境的软件核心是一套交叉编译工具链，运行在本地宿主平台上，它是用于编译生成在目标平台上运

行的可执行文件的一组编译软件。嵌入式交叉开发的本地宿主平台通常采用 X86 Linux 平台。嵌入式交叉开发的目标平台取决于其处理器的体系结构和采用的操作系统。

2) Bootloader

Bootloader 是在系统加电后嵌入式操作系统运行之前执行的一段小程序。它的作用是初始化硬件设备、建立内存空间的映射表，从而建立适当的系统软硬件环境，为调用操作系统内核做准备。

3) 嵌入式 Linux 内核

嵌入式 Linux 的开发和研究是操作系统领域中的一个热点，目前已经开发成功的嵌入式系统中，约有一半使用的是 Linux。Linux 之所以能在嵌入式系统市场上取得如此辉煌的成果，与其自身的优良的特性是分不开的。如：广泛的硬件支持，内核高效稳定，开放源码，软件丰富优秀的开发工具，完善的网络通信和文件管理机制。

4) 根文件系统

根文件系统是 Linux 系统的一个重要组成部分。它提供 Linux 内核运行所必须的库文件、设备文件、系统配置文件等。嵌入式 Linux 支持的文件系统有数十种之多，其中 Yaffs 是一种针对 NAND Flash 存储器开发的嵌入式 Linux 文件系统。根据嵌入式 Linux 系统的特点和所选用的硬件平台，论文采用 Yaffs 根文件系统。

2.4.2 应用软件的设计

应用软件的设计包括：视频采集编码模块设计和视频传输模块设计。

1) 视频采集模块设计：使用 USB 摄像头进行视频采集，需要在嵌入式 Linux 内实现对 USB 驱动的添加，通过 V4L2 编程来实现视频的采集。

视频信号采集不采用芯片级设计方案，而是基于市场上常见的 USB 摄像头采集数据，并通过开发板提供的 USB TYPE A 型接口进行通讯，将视频数据输入 S3C6410 开发板中，进行后继处理。视频信号采集的硬件结构见图 2.5 所示。

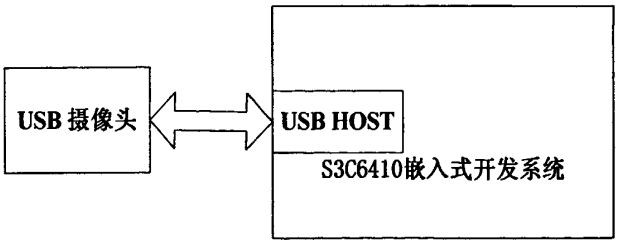


图2.5 视频采集硬件结构图

USB 摄像头属 USB 设备，它将摄取的数字视频图像直接通过 USB 接口送入开发板进行处理。嵌入式系统最常用的摄像头有两大系列，分别是基于 OV511 和 ZC0301 芯片的。其中以 ZC0301 为图像处理核心芯片的摄像头所获取的图像质量更高，在市场上的

应用也更为广泛。故本系统选用以中星微公司的ZC0301为核心芯片的摄像头。

2) 压缩编码软件的选择

H.264/AVC标准自2003年3月开始公布以后，世界各国的视频开发人员很快地开发出符合H.264/AVC标准的多个版本的视频程序，同时很多研究组织公布了他们各自的开源代码。其中有3个开源组织开发的H264标准的程序较为著名，分别是德国的JM版本、法国的X264版本及发源于中国的T264版本，这三大系列具有不同的特点：

JM[8]系列是H.264/AVC的官方测试源码，由德国hhi研究所(Heinrich-Hertz-Institut)负责开发，实现了H.264/AVC所有的特性。由于是官方的测试源码，所以学术研究的算法一般在JM基础上实现并和JM进行比较。但JM程序结构冗长，只考虑引入各种新特性以提高编码性能，忽视了编码复杂度，因此其编码复杂度极高，不宜实用。

X264[9]是网上自由组织联合开发的兼容H.264/AVC标准码流的编码器，是由法国巴黎中心学校的中心研究所(Cenrtale Résuax)的一些学生发起的。X264在程序结构和算法方面较JM系列有较大改进，并利用了MMX/SSE/SSE2等基于X86架构的多媒体硬件指令加速技术，同时摒弃了一些耗时但对编码性能提高微小的模块。和JM相比，X264更注重实用，在不明显降低编码性能的前提下，努力降低编码的计算复杂度。

T264[10]是由中国视频编码自由组织联合开发的H.264/AVC编解码器，编码器编码输出标准的H.264/AVC码流，但解码器只能解T264编码器生成的码流。和X264的出发点相似，T264比较注重实用，吸收了JM、X264、XVDI(MPEG-4)的优点。

X264和JM相比，在编码性能和JM相当甚至更好的情况下，编码速度可以提高300～500倍。T264和JM相比，虽然编码速度有了很大的提高，但其解码器只能解码T264编码的码流，通用性不佳。另外，有测试结果[11]显示，在客观质量(PSNR)相同的情况下，X264的主观图像质量明显比T264的高，而X264在客观质量不如JM86的情况下，有时X264的主观图像质量却比JM86好，这说明X264更加注重实效。因此，通过对码率、失真度、编/解码时间、功能实用性、程序复杂度、通用性等各项因素的测试和比较，选择X264作为本课题的参考模型。

由于该编码技术较为复杂，需要对H.264编码技术进行核心算法的改进以降低复杂度并针对S3C6410平台对编码器进行优化。

3)视频传输模块设计：主要实现将采集压缩处理过后的视频流进行无线发送，网络协议栈使用3G模块内嵌的TCP/IP协议栈，传输层采用RTP/RTCP/UDP协议，通过socket网络通信完成视频传输。

2.5 本章小结

本章首先介绍了嵌入式无线视频监测系统的基本框架设计，然后重点以监测系统部分分别从硬件和软件两方面进行了具体的设计和构架，在硬件部分主要介绍了 ARM11 处理器 S3C6410 平台和 3G 模块；软件框架主要从嵌入式系统软件的构建和应用软件两方面进行了描述。

第三章 嵌入式系统软件开发与移植

Linux从1991年问世到现在，近二十年的时间已经发展成为功能强大、设计完善的操作系统之一，不仅可以与各种传统的商业操作系统分庭抗争，在新兴的嵌入式操作系统领域内也获得了飞速发展。采用对标准Linux小型化裁剪处理之后的嵌入式Linux，可以利用开源代码的易得性、POSIX兼容性、坚实的可靠性以及大量的应用程序，且无需支付任何许可费用。所有这些优点,再加上价格优势，使得Linux成为嵌入式操作系统的一个出色的解决方案。

3.1 嵌入式 Linux 操作系统

嵌入式系统是以应用为中心，以计算机理论为基础，软件硬件可裁剪，适应系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。一般的嵌入式系统具有以下特征：系统内核小；专用性强；系统可裁剪以及实时操作系统(RTOS)的需求。

嵌入式操作系统EOS(Embedded Operating System)是一种支持嵌入式系统应用的操作系统软件，它是嵌入式系统中(包括硬、软件系统)极为重要的组成部分，通常包括与硬件相关的底层驱动软件、系统内核、设备驱动接口、通信协议、图形界面、标准化浏览器等。与通用操作系统相比较，嵌入式操作系统具有在系统实时高效性、硬件的相关依赖性、软件固态化以及应用的专用性等方面的特点。

目前广泛应用的产品包括VxWorks、WinCE、VRTX、PalmOS、PSOS等，这些RTOS都是商用的嵌入式操作系统，它们在系统可靠性和对用户的技术支持都有优势。但是缺点是价格昂贵，核心源代码不公开，可移植性差，难以实现嵌入式系统要以最小的软硬件系统，最低的成本去完成目标功能这一特点。Linux系统与UNIX系统兼容，开放源代码。它原本被设计为桌面系统，现在广泛应用于服务器领域。而更大的影响在于它正逐渐的应用于嵌入式设备。另外，Linux操作系统源码全部公开，任何人可以修改并在GNU通用公共许可证下发行。

3.2 嵌入式 Linux 开发环境的建立

3.2.1 硬件环境

PC宿主机通过串口COM1、并口、SDIO接口、以太网络接口与嵌入式开发板系统的串口、JTAG以及以太网接口分别连接。其中各接口如下：

1) 以太网络接口----传送文件

2) SDIO(SD卡)----烧写Bootloader(Uboot)

3) 串口----传送文件和调试

4) USB转接口----烧写内核及文件系统（速度快）

3.2.2 交叉开发环境的建立

通常开发嵌入式Linux系统宿主机/目标平台构架选用连接式设置，即宿主机和目标平台一直通过交叉线连接在一起，所有的数据都是通过连接传送。其中，宿主机包含交叉开发环境，目标平台则包含引导加载程序、linux内核、根文件系统。

所谓交叉编译，就是在一个平台上生成另一个平台上的可执行代码，即将在X86上的代码编译为ARM系统可以识别的二进制可执行文件。在进行应用程序开发之前，必须在宿主机上建立和配置好开发应用程序所需要的开发环境。

一个完整的交叉编译器包括以下几个部分：交叉编译器(GCC)，库文件(Glibc)，调试工具(GDB)，Linux头文件，以及一些常用的操作二进制文件的工具(Binutils)。由于Linux内核对编译器有很大的依赖性，所以不同版本的Linux需要相应版本的编译器编译。本文采用cross-4.4.1版本编译器，该版本比较稳定。

首先启动PC机Linux系统，拷贝cross-4.4.1.tar.bz2文件到/usr/local。然后安装交叉编译器。在终端窗口输入以下命令：

```
# mkdir /usr/local/arm
# tar xvzf cross-4.4.1.tar.bz2
# mv 4.4.1 /usr/local/arm
# export PATH=$PATH:/usr/local/arm/4.4.1/bin
```

这样，编译器就安装完毕了。上述命令的含义：

tar xvzf cross-4.4.1.tar.bz2：解压压缩文件包。解压完成后，输入ls，可以看到新的目录“4.4.1”。

mv 4.4.1 /usr/local/arm/：移动整个4.4.1目录到 /usr/local/arm/目录下。

export PATH=$PATH:/usr/local/arm/4.4.1/bin:设置系统环境变量。如果输入命令arm-linux-gcc -v后能有版本显示则表明交叉编译环境已经建立好了。完成以上步骤，在个人PC机上就完成交叉编译环境的建立。

3.2.3 配置 NFS

网络文件系统（NFS）是为了在不同的系统间使用文件，所以它的通讯协定设计与主机及操作系统无关。当使用者想用远端文件时只要用“mount”即可把服务器文件系统安装在自己的文件系统之下，使得远端文件使用上和本地机器的文件一样[12]。

在嵌入式Linux的移植过程中，NFS服务主要用于目标系统和主机系统共享相同的文件目录，这样就不再需要上传和下载，直接就可以看到并行编译好的应用程序。

配置NFS服务器，可以通过如下操作来完成：

(1) 主机使用操作系统Redhat 9.0，目标板运行ARM-Linux操作系统，并接入局域网，IP分别为10.0.0.1和10.0.0.2。

(2) 在主机端以root用户运行setup，system service选项中选择NFS，退出setup，然后修改Linux主机上的/etc/exports文件，添加一行：/home/nfs(rw)。保存并退出，然后启动NFS服务service portmap start(portmap为启动NFS服务必须) 、service nfs start 。可以通过PC机自己mount自己，看是否成功就可以判断NFS是否配置好。

3.3 Bootloader 的设计和实现

Bootloader就是在操作系统内核运行之前执行的一段小程序。通过这段小程序，完成初始化硬件设备、建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核准备好正确的环境[13]。图3.1是一个典型的嵌入式应用系统固态存储设备的空间分配示意图。

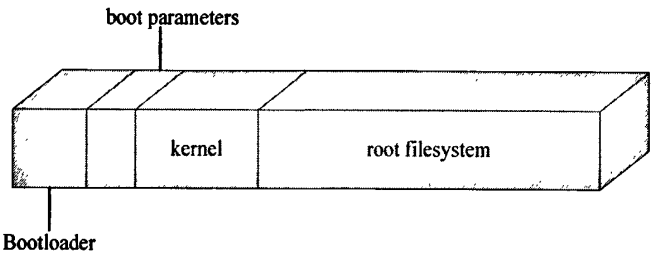


图3.1 Bootloader以及系统各个部件所处的层次

目前大多数Bootloader都包含两种不同的操作模式："启动加载"模式和"下载"模式。

启动加载模式也称为"自主"模式。即Bootloader从目标机上的某个固态存储设备上将操作系统加载到RAM中运行，整个过程并没有用户的介入。

下载(Downloading)模式：在这种模式下，目标机上的Bootloader将通过串口连接或网络连接等通信手段从主机下载文件。从主机下载的文件通常首先被Bootloader保存到目标机的RAM中，然后再被Bootloader写到目标机的FLASH类固态存储设备中。

Bootloader的启动过程可以是单阶段的，也可以是多阶段的，通常Bootloader是两个阶段启动，stage1和stage2两部分。下面是常规的Bootloader设计流程：

Stage1通常包括如下步骤：

1) 硬件设备初始化；

2) 为加载Bootloader的stage2准备RAM空间；

3) 拷贝Bootloader的stage2到RAM空间中；

4) 设置好堆栈；

5) 跳转到stage2的C入口点。

Stage2包括如下步骤：

1) 初始化本阶段要使用到的硬件设备；

2) 检测系统内存映射；

3) 将kernel映像和根文件系统映像从flash上读到RAM空间中；

4) 为内核设置启动参数；

5) 调用内核。

Stage1通常用汇编设计，主要完成：硬件初始化，为加载Bootloader的Stage2准备内存空间，复制Bootloader的Stage2到内存中，设置堆栈，跳转到Stage2的C入口。

Stage2常用C语言来设计，可以实现复杂的功能，并具有更好的可移植性。该阶段主要完成：初始化本阶段要用到的硬件，检查系统内存映射，将内核映像和根文件系统从Flash复制到内存中，为内核设置启动参数，加载内核。

Bootloader是严重依赖于硬件而实现的，所以在嵌入式开发中建立一个通用Bootloader几乎是不可能的，通常也都需要对已有的Bootloader进行移植工作，如armboot、blob、vivi和uboot等。本系统设计完成了Uboot代码在S3C6410平台的移植工作。

Uboot是ARM-Linux上常用的Bootloader程序，它结构简单，功能完整，支持xmodem、tftP传输，Flash编程和分区；支持Linux内核引导，可传递内核参数。Uboot的启动过程也分为两个阶段：stagel和stage2。Uboot的这两个部分都有独立的连接脚本，Makefile对它们分别编译生成elf格式和bin映象文件，再通过dd命令把这两部分映象复制到一个文件中，得到最终的可直接运行的映象文件。

Stagel的连接脚本如下：

```
OUTPUT_FORMAT(“elf32-littlearm”，“elf32-littlearm”，“elf32-littlearm”)
OUTPUT_ARCH(arm)
ENTRY(_start)
SECTIONS
{
    .=0x00000000;
    .=ALIGN(4);
.....
}
```

脚本设置了Uboot代码是从0x0开始的，入口由_start符号指定。

Stage2部分的运行地址，通常在内存中，所以才可以实现对Flash空间的编程。而不

同的arm平台有不同的定义，本系统使用的平台是0xa0000000。

对于ARM-Linux内核，不管是压缩内核还是非压缩内核，要求在跳到内核时满足如下条件：

1) ARM寄存器设置：R0=0；R1=平台类型ID(参考/linux/arch/tools/mach-types)；R2=启动参数标记列表在内存中的起始基地址。

2) ARM处理器必须处在超级用户模式(SVC)且禁止中断(IRQ和FIQ)。

3) 数据缓冲DCache和MMU必须关闭。

所以将src/blob/linux.c中的代码配置如下：

```
theKernel=(void(*)(int，int))KERNEL_RAM_BASE(0xA0008000);
theKernel(opt[0]，opt[l]);
opt[0]=0;
opt[l]=200；//表示引导的是S3C6410
```

Bootloader首先把位于flash的Bootloader拷贝到SDRAM的(0xA4000000-0x80000)，也就是SDRAM地址的最高端512K用来放Bootloader。然后把Kernel Image从0x000c0000处搬迁到0xA0008000处。引导内核。

3.4 Linux 内核移植

3.4.1 内核结构

Linux内核主要由5个子系统组成：进程调度，内存管理，虚拟文件系统，网络接口，进程间通信[14]。一般在每个目录下都有一个depend文件和一个makefile文件。这两个文件都是编译时使用的辅助文件。其中makefile文件中指出的编译时需要用到的编译器，也是移植过程中不可缺少的。

1) arch目录

Linux系统可以支持如此多的平台部分原因是由于内核把源代码划分为体系结构无关部分和体系结构相关部分。arch目录包含了体系结构相关部分的内核代码。其中每个目录都代表一种硬件平台，比如本文使用的ARM平台和PC使用的i386。对于任何一个平台，都必须包括以下一个目录。

boot：包括启动内核所使用的部分或全部平台特有的代码。

kernel：存放支持体系结构特有的特征的实现。

lib：存放高速的体系结构特有的通用函数的实现。

mm：存放体系结构特有的内存管理程序的实现。

math-emu：模拟FPU的代码，对于ARM处理器来讲，此目录用mach-xxx代替。由此

可知，移植的重点就是移植arch目录下的文件。

2) drivers目录

该目录下保存了所有的设备驱动程序。其源码占整个内核发行版代码的一半以上。有些驱动程序是与硬件平台无关的而有些是相关的。

3) fs目录

该目录下列出的Linux支持的所有文件系统。目前Linux已支持包括NTFS在内的多种文件系统。

4) include目录

该目录包含编译核心所需要的大部分头文件，如与平台无关的头文件在include/Linux子目录下。不同的平台需要的头文件有所不同，故该目录和arch目录一样，按平台划分了多个子目录，如asm-arm目录等。

5) init目录

init目录下包括核心的初始化代码，有main.c和version.c两个文件。

6) ipc目录

Ipc目录包括了核心的进程间的通信代码。

7) kernel目录

内核管理的核心代码在这里，在与处理器结构相关的代码都放在arch/*/kernel目录下。

8) lib目录

该目录包含于平台无关的诸如strlen和memcpy之类的通用函数。

9) mm目录

该目录包含了所有的内存管理代码，与具体硬件体系结构相关的内存管理代码位于arch/*/mm目录下。

10) net目录

该目录是核心的网络部分代码，其每个子目录对应于网络的一个方面。

11) 其他目录

还有一些目录如documentation和scripts目录，documentation目录存放文档目录。Script目录主要在配置是用到，包含内核的一些脚本文件。

3.4.2 内核的修改、配置和编译移植

对Linux内核的移植修改，主要集中于与体系结构相关的部分。在arch目录中可以看到许多目录，常以芯片命名，表示针对该芯片体系结构的代码。针对ARM核的S3C6410处理器需要修改和添加的文件集中于linux/arch/arm/mach-s3c6410和

linux/include/asm-arm两个目录下。其中linux/arch/arm/下其他的目录包含通用的ARM的代码：kernel(核心代码)、mm(内存管理代码)、lib(ARM特殊的或经过优化的内部库函数)、nwfpe fastfpe(两种不同的浮点实现)、boot(最终编译过的内核所在目录，包含生成压缩内核的工具)、def-configs(包含给每种机器默认的配置)；linux/include/asm-arm/下是一些通用目录：arch(链接到已配置机器的头文件arch-pxa)、hardware(与ARM有关的芯片及设备的头文件)、mach(很多机器使用的通用的接口定义和机器描述符宏)、proc(根据配置的机器的情况连接到proc-armo和proc-armv)、proc-armo和proc-armv(ARM内核处理器的头文件的26位和32位版本)。

ARM-Linux的板级移植过程如下：

首先准备一些必要的文件。主要修改的内容是内核入口部分(head-armv.S)，处理器和体系结构相关的部分，I/O端口映射部分和中断初始化部分(entry-armv.c)。需要修改的部分有：

1) 登记machine-ID。将S3C6410 CONFIG_CPU MACH-TYPE_S3C6410 200添入到linux/arch/arm/tools/mach-types文件中。脚本linux/arch/arm/tools/gen-mach-types需要linux/arch/arm/tools/mach-types来产生linux/include/asm-arm/mach-type.h文件(编译的时候产生)，这个文件设置了一些定义和宏。源代码使用这些宏和定义来选择合适的汇编代码(操作系统中会根据不同的机型设置不同的代码，而提供的接口是一样的)。

2) 配置文件。linux/arch/arm/def-configs包含以<machine-name>命名的默认的配置文件。编辑linux/arch/arm/config.in文件以便make config会支持S3C6410。这个文件说明了机器的CONFIG_symbols以及与其他CONFIG_symbols之间的依赖关系。其实这个文件是在后面的make menuconfig中自动产生的。在这里可看到机型的各种配置。另外，还需要加入一些对外围设备的支持选项。

3) Makefile文件。插入下面的语句到Arch/arm/Makefile，表明该机型采用的是S3C6410

```
ifeq($(CONFIG_ARCH_S3C6410)，y)
MACHINE = S3C6410
endif
```

Arch/arm/boot/Makefile

在这个文件中定义ZTEXTADDR(内核解压后开始的地址)，这个地址值应和Bootloader中设定的Linux被加载的地址相同。标准的做法是在RAM基址以上32K(8000)处，RAM基址和内核开始地址之间的空间留给页表使用。

```
ifeq($(CONFIG_ARCH_PXA)，y)
```

ZRELADDR=0xa0008000 //内核开始的地址是0xa0008000

endif

4) 中断初始化和中断处理部分。arch/arm/kernel/entry-armv.S中提供汇编语言编写的宏disable_fiq、get_irgnr_and_base、irq_prio_table。disable_fiq和irq_prio_table通常是空的，get_irgnr_and_base用来从处理器中取得中断号。

5) 调试函数的设置。arch/arm/kernel/debug-armv.S里的函数不依赖中断或者任何内核函数，如果内核不能启动，需要这些函数来调试内核，addruart(设置UART的地址)、senduart(发送)、waituart(等待)、busyuart(UART忙)。

6) 设置有关开发板的体系结构。arch/arm/mach-pxa/Makefile为机器添加目标代码，在这个文件中列出目标文件。

_TARGET：=S3C6410.o

Obj-y+=generic.o irq.o dma.o

7) 其他设置。完成以上步骤后，还需要修改和系统时间以及内存映射等有关的文件。修改完相关文件后，就可以通过make menuconfig、make dep、make zImage配置内核，编译内核，生成内核映像文件，可以下载到系统中去。

如果上述步骤全部正确，就会产生可引导的压缩内核，并且通过调试函数输出调试信息。但是，还需要添加支持外围设备的程序。相关文件的修改在linux/drives(驱动所在的目录)Linux/fs(文件系统目录)Linux/net(网络协议目录)。

以上内核移植过程可总结为：移植的过程中需要修改相应的体系结构有关的文件和初始化中有关的文件。初始化部分包括芯片级的初始化，这部分主要和芯片硬件相关，可以直接从芯片厂商的开发手册中得到相应的代码段。然后是板级的初始化，这部分主要是特定板子的初始化和相应外部设备驱动程序的开发，是移植的重点。

3.5 Linux 根文件系统设计与实现

标准的Linux中一些文件虽然功能强大，但体积过大，在嵌入式Linux中，受存储器的限制，不能使用标准Linux那样复杂的根文件系统，需对文件系统进行重新设计和裁剪。一般嵌入式linux文件系统需要的目录如表所示：

表3.1 嵌入式Linux文件系统目录

目录名	作用	是否可选
/bin	可执行程序	必选
/boot	启动LINUX时使用的核心文件	可选
/dev	接口设备文件目录	必选
/etc	系统管理所需要的配置文嘉和子目录	必选
/home	一般用户的主目录	必选
/lib	系统最基本的动态链接共享库	必选
/mnt	用户临时挂在别的文件系统	可选
/opt	其他安装的软件包	可选
/proc	一个虚拟的目录，它是系统内存的映射，可通过直接访问这个目录来获取系统信息，这个目录的内容不再硬盘上而在内存中	必选
/root	这个目录为超级用户的主目录	可选
/sbin	系统管理员使用的系统管理程序	必须
/tmp	这个目录是用来存临时文件的	可选
/usr	用户程序	可选
/var	这个目录下存放着在不断扩充的东西，包括各种日志文件	可选

1) 嵌入式Linux文件系统内容实现具体过程如下：

(1) 在/dev目录下创建需用到的Linux系统中使用的外部设备；

(2) 在/etc目录下创建系统相关配置文件；

(3) 在/lib目录下建立需要的函数库；

(4) 在/bin和/sbin目录下，利用Busybox建立基本命令工具程序和系统管理程序。

Busybox是一个多调用的二进制文件，它提供的POSIX的Linux命令最小子集，其大小仅100K左右，适合于非常小的嵌入式系统。它包括ash shell，ifconfig，halt，reboot，mkdir，mount，ls，echo，cat等一些系统上常用的工具程序，且使用busybox很简单。虽然与相应的GNU工具比较起来，Busybox所提供的功能和参数略少，但在嵌入式系统中，它足以满足[15]。

2) 嵌入式Linux根文件系统类型设计与实现

Yaffs文件系统是专门为NAND闪存设计的文件系统，建立在MTD(menory

technology device)基础上。MTD是Linux上为访问Flash设备而开发的一个抽象接口，目标是在硬件驱动程序和上层程序之间提供通用接口，提供一组对底层闪存系统进行类似于标准文件read，write和erase等操作的统一接口。通过这个接口，就可以像读写普通文件一样对Flash设备进行读写操作。

制作YAFFS映像。首先下载yaffs文件系统的制作工具：mkyaffsimage，执行如下的命令即可生成所要的映像：

Chmod 777 mkyaffsimage

//设置mkyaffsimage为可执行权限，即mkyaffsimage成为可执行文件

执行以下命令生成根文件系统的映像文件：

mkyaffsimage /root/myrootfs myrootfs.yaffs

//生成了名为myrootfs.yaffs的映像文件，这样文件系统的移植也已完成，最后将文件系统下载到系统特定的地址中。至此我们完成了S3C6410开发板上Linux操作系统的全部移植过程。

3.6 本章小结

本章首先介绍了嵌入式Linux的发展与特点，具体给出了交叉开发环境的建立方法，然后分析了Bootloader的启动过程，给出了Bootloader的移植设计，接着通过修改内核文件完成了内核的移植，最后对根文件系统进行设计并实现移植工作，至此形成了一个运行正常的嵌入式Linux系统环境，为后继的应用软件开发打下了良好的基础。

第四章 H.264/AVC 编码结构及关键模块算法优化

H.264/AVC[16]编码器编码框架与以往的编码标准中的编码器模块没有太大的区别，它的混合编码方案仍然采用预测与变换相结合的方法。差别是，H.264/AVC在多个功能模块上均引入了一些新的关键技术，使得各个模块的细节在具体实现上有了较大的改变，通过这些改变，在相同的编码框架下，新的标准获得了更高的视频压缩性能和更广的适用性。

4.1 H.264/AVC 编码流程

从图4.1中可以看出，H.264/AVC编码器包括两个数据流路径：前向路径(从左向右)和重构路径(从右向左)。

对于前向路径，F_n为编码帧，该帧以宏块(原始图像中16×16点阵)为单元被编码器处理。每一个宏块被编码成为帧内或者帧间模式。无论处于哪一种模式，一个预测宏块P被基于重构的帧所构建。在帧内模式中，P从当前已经编码过、解码过和重构的帧F_n的采样所获得。在帧间模式中，P通过源自一个和多个参考帧的运动补偿预测中形成。图中，参考帧表示为先前编码帧F'_{n-1}；然而每一个宏块的预测可能从一个或者两个先前的或者未来帧(按照一定的时序)所形成。

预测值P与当前块相减得到一个残差块D_n；然后将D_n经过DCT变换和量化产生一组变化系数X，再对X进行重排序和熵编码。熵编码的系数，随同用来解码宏块的边信息形成压缩的比特流。该比特流传送给网络抽象层(NAL)用来传输或者存储。

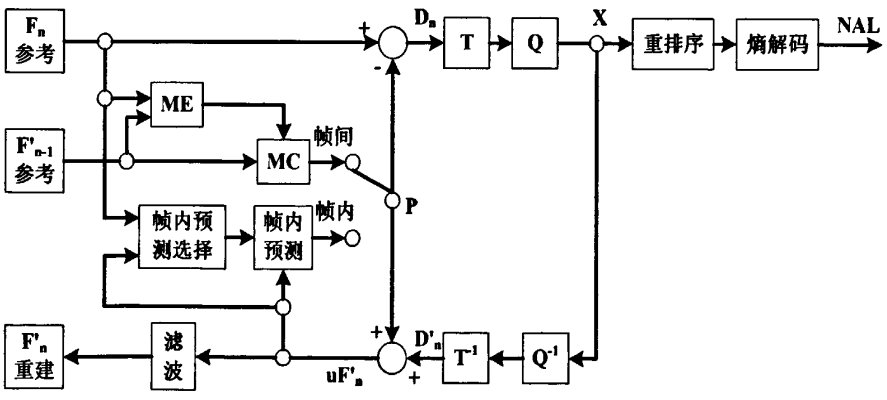


图4.1 H.264/AVC编码器原理图

对于重构路径，对量化变换系数X进行与解码器解码相同的反量化和反变换过程，导出解码预测残差D_n'，(这与原来的残差块D_n并不完全相同，因为量化和反量化的过程中产生了信息的耗损)，解码预测残余D_n'与预测P相加，得到重建的宏块uF_n'，送到去块

效应滤波器，得到当前重建帧F_n'，同时将其存储以供将来做后续帧的帧间预测模式的参考帧。

H.264/AVC性能的提高是靠多个模块共同作用的结果，下面以x264为参考模型介绍H.264/AVC的一些核心基本模块，同时x264也是将要进行移植和优化的原始模型。

4.2 X264 编码参考模型及 H.264/AVC 关键模块分析

4.2.1 X264 编码参考模型

X264编码模型的具体实现可以分为由高到低的五个层次：序列组→序列→帧→条带→宏块。

一个视频往往是由多个序列构成的，它的最高层，称为序列组层。一般来说，每个序列代表一个场景，一个序列内部的帧拥有大多数相同的编码参数。而不同的序列/场景之间的编码参数可能有较大的不同。

1) 序列组层

在X264的程序中，最高层为main函数及编码功能主函数Encode函数，通过循环调用x264_encoder_encode函数进行图像序列的编码。特别地，为了适应分组交换网的传输，编码后的码流需要打包成NALU(Network Abstraction Layer Unit)。如下图4.2所示：

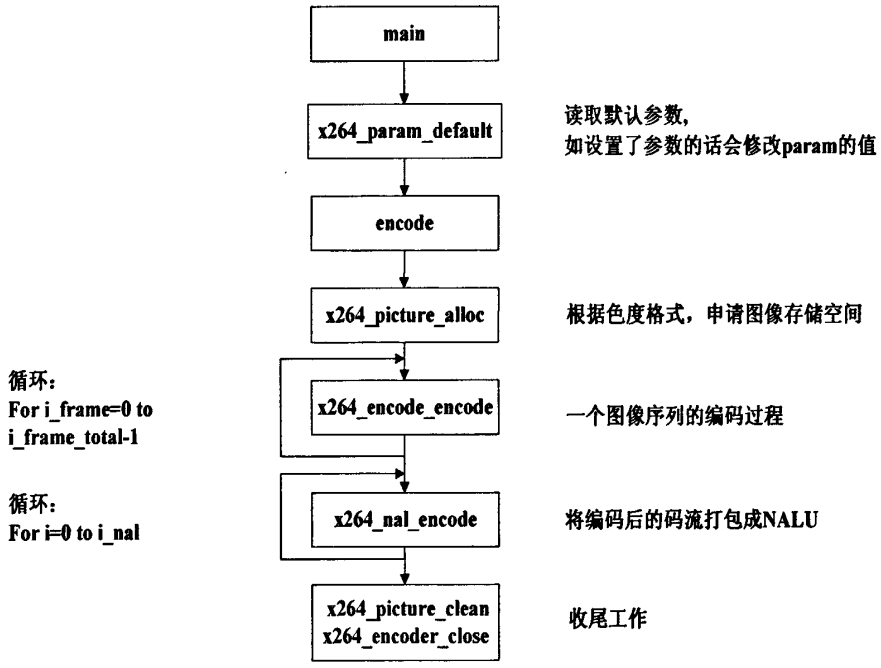


图4.2 Encode函数流程

2) 序列层

x264_encoder_encode函数为编码算法的第二层，它通过循环调用x264_slice_write函数对一个序列/场景进行编码并进行NAL层处理。由于原始视频序列的帧顺序(即显示顺

序)和编码顺序是不同的，且B帧需要双向的参考，于是需先编码B帧之后的非B帧，即：在不对P帧进行编码之前，暂不对B帧进行编码，只是把B帧放进缓冲区(采用frame = x264_encoder_frame_put_from_picture(h, h->frame_next, pic)函数将这B帧放进h->frame_next中)。如视频序列I B B P B B P，先确立第一个帧的类型 - I帧，然后对I帧编码。接下来是2个B帧，将它们放进缓冲区数组，然后是P帧，这时判定它的类型并进行编码。同时缓冲区的B帧放进h->bframe_current[i]，不过这时P帧前的两个B帧并没编码，当读到P帧后面的第一个B帧时，才将h->bframe_current数组中的第一个B帧编码。图4.3为x264_encoder_encode函数的帧编码循环：

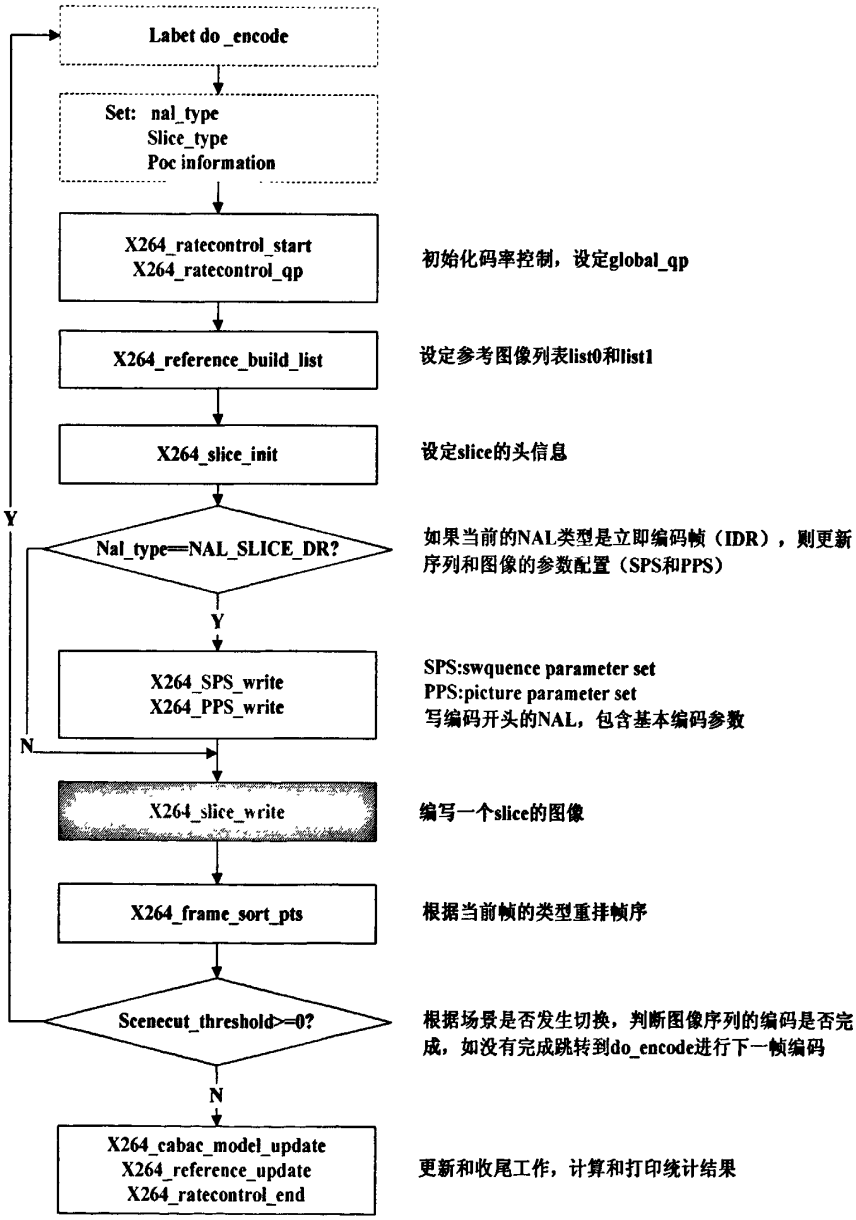


图4.3 x264_encoder_encode函数流程：帧编码循环

3) 帧和条带层

x264_slice_write 函数为编码算法的第三层(如图 4.4 所示)，用循环逐个对宏块调用 x264_macroblock_analyse 进行运动估计(ME)，保存运动矢量；调用 x264_macroblock_encode 进行运动补偿(MC)，计算残差，完成对一个 Slice 的编码，同时重建与解码端同步的参考帧 f_dec。

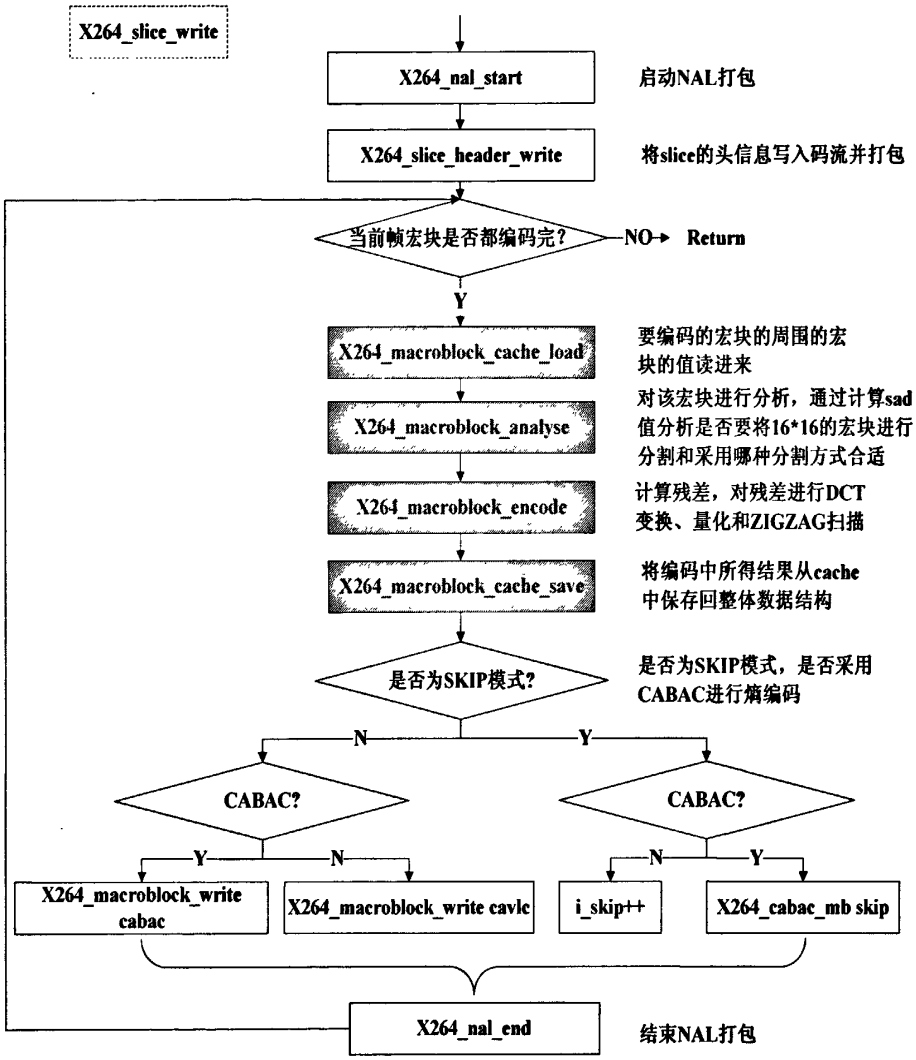


图4.4 x264_slice_write函数流程

4) 宏块层

x264_macroblock_analyse 和 x264_macroblock_encode 函数为编码算法的最低层。x264_macroblock_analyse 计算周围的宏块的值，分析各种可能运动预测模式下的 cost，寻找到最合适的 ME，保存运动矢量。x264_macroblock_encode 函数对 intra 模式的宏块计算残差，对 inter 模式的宏块先进行运动补偿后计算残差；对残差进行 DCT，量化，ZigZag 扫描排序后，以 4×4 子块参数的形式，保存在若干个一维数组中；原量化后的 DCT 系数矩阵再反量化、IDCT 后得到与解码端同步的重建帧 f_dec，并记录当前的模式为下次编码宏块(亚宏块)做参考。

4.2.2 H.264/AVC 关键模块分析

4.2.2.1 帧内预测

帧内编码是传统混合编码框架的重要组成部分。当对一帧图像进行编码时没有利用任何其它帧图像的信息，则对该图像的编码就称之为帧内编码。帧内编码主要是通过消除图像的空间冗余信息来实现对图像的压缩。

以往的标准中只采用帧内编码的图像称为 I 图像，在 I 图像中是直接把像素块的数值进行变换。这样处理的结果就导致 I 图像包含了大量的信息，压缩效率不是很高。H.264/AVC 根据相邻像素可能相同的性质，利用了相邻像素的相关性，采用了新的帧内预测模式。通过当前像素块的左边和上边的像素进行预测，只对实际值和预测值的差值进行编码，这样就能用较少的比特数来表示帧内编码的像素块信息[17]。

在 H.264/AVC 标准中，对于亮度样值，预测块 P 可以有 4×4 和 16×16 两种尺寸。对于 4×4 的亮度块，一共有 9 种可选的预测模式。对于 16×16 的亮度块，有 4 种选择模式。对于色度分量，也有 4 种选择模式[18]。

1) 4×4 的亮度块预测

如表 4.1 所示，4×4 亮度块的上方和左方像素 A～Q 为已编码并重构的像素，用作编码器的预测参考像素。A～p 为待测的像素，利用 A～Q 的值和 9 种预测模式实现。其中模式 2(DC 模式)的预测根据 A～Q 中已编码像素进行预测，而其余模式只有在所需预测像素全部提供后才能使用。

表4.1 预测标记点

Q	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				
M								
N								
O								
P								

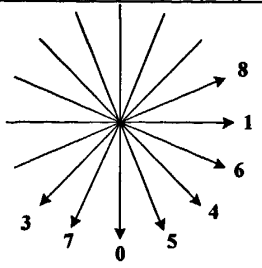


图4.5 4×4亮度的9种预测方向

2) 16×16 亮度宏块的预测

如果视频图像区域内部较为平坦，图像细节不多，此时可以对 16×16 的像素块直接进行帧内预测编码。16×16 像素块的帧内预测共有四种预测模式，分别是：垂直预测，水平预测，DC 预测和平面预测，具体如图 4.6 所示：

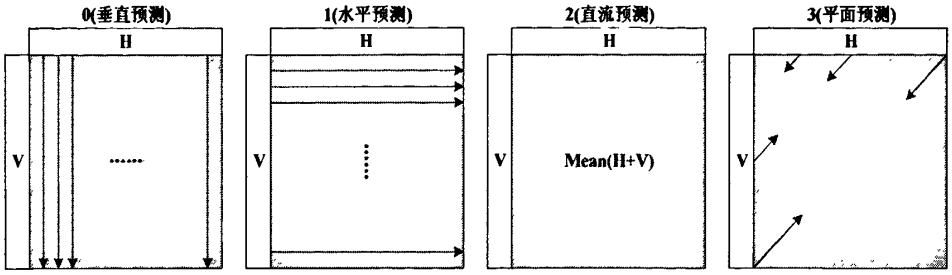


图4.6 4种16×16帧内预测模式

3) 8×8 色度预测

帧内编码宏块中的两个 8×8 色度块分量都是利用上边或是左边已编码的色度值进行预测的，并且它们总是采用相同的预测模式。其 4 种预测模式和 16×16 亮度预测模式相似。只是模式编号不同，DC(模式 0)，水平(模式 1)，垂直(模式 2)，平面(模式 3)。

X264 程序是基于率失真函数来进行预测模式选择的，计算公式如下：

$$J(s,c,MODE|\lambda_{\mathrm{mode}}) = SATD(s,c,MODE|QP) + \lambda_{MODE} \times R(s,c,MODE|QP) \quad (4.1)$$

其中，QP 代表宏块编码时的量化参数，λ_{mode} 是模式选择时的拉格朗日系数，MODE 是代表众多预测模式中的一种，SATD 是原始宏块 s 和预测宏块 c 之间差值变换的绝对值之和，R(s，c，MODE|QP)代表选择 MODE 预测模式时编码该宏块所需要的比特数。通常把 J(s，c，MODE|λ_{mode})的值称为率失真开销。所谓最优模式，就是预测块具有最小率失真开销时该块的预测模式。

X264 采用的帧内预测选择算法流程如下：

(1) 根据相邻宏块的存在情况，判断可用的 16×16 帧内预测模式；

(2) 计算每种 16×16 帧内预测模式下的率失真开销，并记录最小率失真开销 i_sad_i16×16 和最优 16×16 帧内预测模式 i_predict16×16；

(3) 若该宏块属于 P 条带，且 i_predict16×16>帧间模式的最小率失真开销 i_cost_inter 的两倍，则不进行其他帧内预测模式的判断，转(9)；

(4) 将宏块划分为 16 个 4×4 的子块；

(5) 根据相邻块预测每个子块的模式。以上方和左方子块 A、B 的模式的最小值作为当前子块 E 最可能的预测模式。如果 A 或 B 的预测模式不可用(在当前 slice 外或不是 4×4 模式)，则把 A 或 B 认为是模式 2(DC 预测模式)；

(6) 根据相邻宏块的存在情况，判断可用的 4×4 帧内预测模式；

(7) 分别对每个子块进行 4×4 帧内预测，计算各 4×4 帧内预测模式下的率失真开销。由于 H.264 的码流结构，编码预测模式所需要的比特数比编码预测模式所需要的比特数要少；

(8) 记录每个子块的最优 4×4 帧内预测模式 i_predict4×4，并将 16 个子块的最小率失真开销相加，得到 i_sad_i4×4；

(9) 比较 i_sad_i16×16 和 i_sad_i4×4，若 i_sad_i16×16< i_sad_i4×4，则采用最优 16×16 帧内预测模式 i_predict16×16，否则采用最优 4×4 帧内预测模式。

4.2.2.2 帧间预测

H.264 帧间预测是利用已编码视频帧和基于块的运动补偿的预测模式。视频中的图像是有连续的帧组成的图像序列，由于景物变化速度的限制，相邻帧之间存在很高的相关性，即存在很高的时间和空间冗余。如何消除这些冗余则成为了视频编码的关键技术，帧间预测就是要消除时域冗余。和以往的视频压缩编码标准相比，H.264 在帧间预测方面采用了很多新的技术，主要包括树状运动补偿、亚像素运动估计、多参考帧预测模式的使用。

1) 树状结构运动补偿

H.264/AVC 支持运动补偿块的大小从 16×16 到 4×4 的亮度采样两者之间的多个选择。每一个宏块（16×16 采样）的亮度分量可按下图 4.7 的 4 种方式进行分割：16×16，16×8，8×16 或者 8×8。

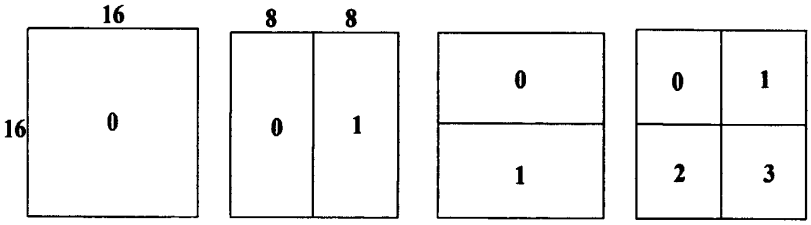


图4.7 宏块划分16×16，16×8，8×16或者8×8

每一个子划分区域是一种宏块的划分。如果选择 8×8 模式，在宏块中可以进一步划分为 4 种方式，如图 4.8：8×8，8×4，4×8 或者 4×4。

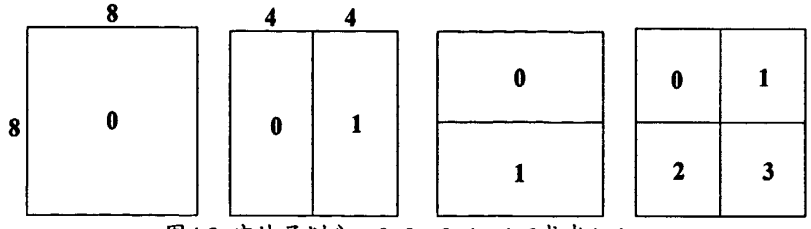


图4.8 宏块子划分：8×8，8×4，4×8或者4×4

一般而言，块越大，选择对应模式的概率越大，由于帧间各模式间具有冗余，可以利用这种相关性，根据先得到的模式的开销值，粗略估计模式的选择范围，跳过对某些

不太可能的模式的检查计算。

2) 亚像素运动估计

帧间编码宏块的每一个分块都是由参考帧中相同大小的区域预测得到。这两个区域之间的偏移量即运动矢量。运动估计就是利用视频图像的时域相关性，产生相应的运动矢量，尽可能准确的描述对象的时域运动。因此运动矢量的精度越高，运动估计的残差越小，这样在降低编码码率的同时提高重建视频的质量。从 H.261 到 H.264/AVC，运动矢量的精度也从整像素提高到 1/4 像素。H.264/AVC 支持亮度分量的 1/4 像素和色度分量的 1/8 像素的运动估计，并详细的定义了相应亚像素的插值实现算法，利用 6 抽头滤波器产生 1/2 分数像素、线性插值产生 1/4 分数像素、4 抽头滤波器产生最高 1/8 分数像素[19]。

3) 多参考帧预测模式

在 H.264/AVC 中，允许采用多个参考帧来进行运动补偿，这在很多自然场景的周期变换以及镜头在两个场景中交替转换等情况下可以提高编码效率。在这种模式下，对某一个块进行运动估计时，编码器会从过去的一个或几个刚编码过的参考帧中选定一帧作为参考帧，以获得更好的预测效果。

4.2.2.3 整数变换与量化

为了进一步节省图象传输码率，需要对图象信号进行压缩，有两种途径可以实现该目标：消除图象像素之间的相关性和减少图象编码的动态范围，即变换编码和量化技术。H.264 标准中采用了基于整数的变换编码技术，大大降低了计算复杂度。量化过程根据图象动态范围的大小来确定量化参数。

1) 整数 DCT 变换

变换编码是图象和视频压缩的重要部分。空域图象像素可以转化到另一个空间，即频域。空间图象像素具有很强的相关性，其能量是均匀分布的，很难被压缩。然而通过某种变换，可以将空域的图象变换到频域中，频域中图象像素的能量聚集在少数的系数上，像素间的相关性大大降低，这样就可以通过去除一些“不重要”的点来进行图象的压缩。两种应用非常广泛的变换方法是：离散余弦变换 DCT[20]和离散小波变换 DWT。DCT 变换通常用于较小的、较规整的图象块，而 DWT 通常用于较大的图象块甚至整个图象。

DCT 变换是由 Ahmed、Natarajan 和 Rao 于 1974 年提出的，因其优越的压缩特性应用越来越广泛。DCT 是目前非常好的变换方法，其性能最接近于统计上最佳的 Karhunen-Loeve 变换，它是正交变换，其产生的系数很容易被量化、算法性能好、具有快速算法、在软硬件中都易于实现、变换是可逆的，以此用来解压缩图象。

然而，以往的DCT变换的变换矩阵存在无理数，在进行DCT变换时需要浮点运算。这样，在编解码的计算过程中由于存在舍入误差，使得反变换的结果和变换前的结果不一致，造成解码后数据的失配，引起漂移。所以，利用有限精度的浮点DCT实现变换编码是有损的，这在一些特定的场合是不允许的，比如医学图象、遥感成象等。此外，在硬件的实现中不同的处理器对浮点数和取整的定义不同也会造成结果的偏差。于是H.264中提出了基于整数的DCT变换，变换的核心运算部分只用到了加法和移位，不需要乘除运算。

2) 量化过程

一幅图象经过帧内或帧间预测编码后形成的残差经过整数DCT变换和Hadamard[21]变换后得到一系列系数，即直流系数DC和交流系数AC，分别代表了图象的低频分量和高频分量。图象的能量主要集中在直流系数上，所以可以利用量化来去除一些高频分量，从而达到数据压缩的目的，而没有任何明显的视觉保真度损失。

量化是在不降低视觉效果的前提下减少图象编码的长度，减少视觉恢复中不必要的信息。通常，输入值的动态范围很大，需要用多个比特数来表示一个数值，经过量化后只能取有限个整数，可用较少的比特数表示。这“有限个整数”就是量化级，每个量化输入被强行归一到与其接近的某个输出上，即量化到某个级。量化处理总是把一批输入量化到一个输出级上，所以量化是多对一的处理过程，是不可逆的，所以存在信息丢失，即存在量化噪声。

4.2.2.4 熵编码

以往标准的熵编码采用变长的霍夫曼(Huffman)编码，码表统一，不能适应变换多端的视频内容，影响编码效率。H.264/AVC根据视频内容不同，采用了两种熵编码的方式，基于上下文的自适应二进制算术编码(CABAC[22])和基于上下文自适应可变长码(CAVLC[23])。

由于在概率估计方面的不同，这两种方法的计算复杂度和压缩效率各有不同。CAVLC根据残差的统计特性，设计了多个码表。在编码时根据语法元素进行码表的切换，具有自适应能力，所以编码效率较高。但是CAVLC也存在着一些缺点，如CAVLC使用的是静态概率估计码表，没有考虑到不同视频流的统计特性，也忽略了符号间的相关性，没有利用相邻符号的信息来服务于当前宏块的编码。这些缺点限制了CAVLC的编码效率，尤其是在高码率下的编码效率更是不理想。

基于上下文的自适应二进制算术编码CABAC则完全克服了CAVLC的这些缺点。CABAC的编码过程主要是：二进制化、上下文建模和二进制算术编码。CABAC中内建了由大量的实验统计而得到的概率模型。在编码过程中，CABAC根据当前所要编码

的内容以及先前已编码好的内容，动态地选择概率模型来进行编码，并实时更新相对应概率模型，并且，CABAC 在计算量和编码速度上进行了优化，用了量化查表、移位、逻辑运算等方法代替复杂的概率估计和乘法运算。因此编码效率较高，在相同编码质量下比 CAVLC 编码节省 10~15%的码率。

4.2.2.5 去块效应滤波器

H.264/AVC 是基于块的编码结构，会有块效应，为尽量消除块变换造成的块效应现象，H.264/AVC 采用了一个自适应的环路滤波器[24]。这个滤波器根据块边缘信息的不同采用不同的滤波权重，因而可以在有效消除块效应的同时又不会影响图像的锐度；另外，环路滤波是作为编码器的一部分直接对编码器端的参考图像进行的，与仅仅作为后处理的解码器端的去方块滤波相比，环路滤波在改善主观质量的同时还可以有效地提高编码器的编码效率。

4.3 H.264 算法复杂度分析

H.264 引入了许多新的编码特性，使编码性能显著提高的同时，既在编码的基本模块中增加了复杂度，也使整体算法实现的复杂度成倍的增加。

对于编码模块的复杂度分布情况，不少学者对此进行了研究，在不同参数配置实验中，帧间预测中需要用到的运动估计始终是计算复杂度最高的部分。实际的具体应用中往往要求大幅度地降低运动估计的复杂度，而运动估计也确实存在较大的优化空间。在一个实时的编码器中，各项技术按照计算量大小排序依次为[25]：运动估计约占整个计算量的 54.8%，4×4 的帧内预测编码占 24.5%，1/4 内插占 9.9%，DCT 变换占 5.2%等等。帧内模式判决主要包括 9 种 4×4 模式和 4 种 16×16 模式的预测选优。由于 H.264 的帧间编码允许使用帧内模式，故帧间宏块模式判决要比较帧内模式的编码效率，因此帧内模式判决也占有不少的复杂度比例。

为使 H.264 编码算法满足嵌入式应用的要求，论文对运动估计算法和帧内预测算法这两部分占复杂度最大的模块进行算法改进和优化。

4.4 运动估计算法

在 H.264/AVC 标准中，引进了更高精度(如亮度上的 1/4 像素精度)的运动矢量和多参考帧进行运动估计，使得在给定的码率条件下可得到尽可能好的编码效率。运动估计是 H.264/AVC 视频编码中非常重要的部分，它有效提高编码性能的同时，由于要进行大量的算法搜索也是编码过程中非常耗时的部分，对这部分进行分析、优化，可以有效地提高编码的速度，降低编码的复杂度，从而更好地满足高效编码的要求。

4.4.1 基本原理和判定准则

运动估计是去除视频中时间冗余的方法。运动估计的实现有多种方法，大体上可以分为四类：递归估计算法[26]，光流估计法[27]、贝叶斯估计算法[28]，块匹配算法[29]。其中基于块匹配的运动估计算法因其算法简单和便于大规模集成电路实现等优点，已被应用于包括 H.264/AVC 在内的主流视频编码标准中。

块匹配运动估计的基本思想是将图像序列的每一帧分成许多互不重叠的块，并认为块内所有像素的位移量都相同，然后对于当前帧中的每一块到前面或是后面的参考帧中某一给定搜索范围内根据一定的匹配准则找出与当前块最相似的块，即匹配块，由匹配块与当前块的相对位置计算出运动位移，所得运动位移即为当前块的运动矢量(Motion Vector，MV)。利用搜索得到的运动矢量在参考帧上进行运动补偿。块匹配基本原理如下图 4.9 所示：

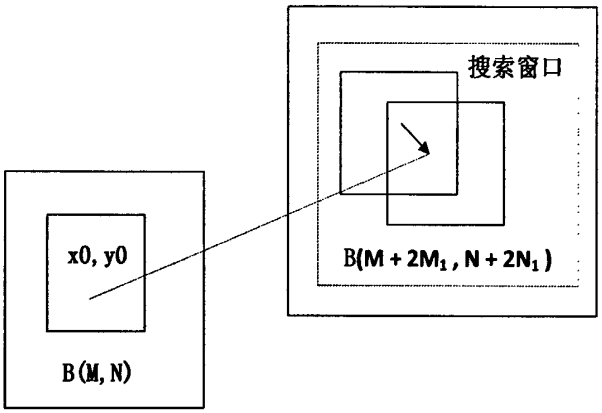


图4.9 块匹配法基本原理

块匹配运动估计算法中依据各种准则函数来衡量匹配的准确度，目前常用的匹配准则有平方绝对误差(MAD)、绝对误差和(SAD[30])、均方误差(MSE)、和归一化互相关函数(NCCF)。

1) 平均绝对误差

$$MAD(i,j)=\frac{1}{MN}\sum_{m=0}^{M}\sum_{n=0}^{N}\left|f_k(m,n)-f_{k-\Delta t}(m+i,n+j)\right| \tag{4.2}$$

2) 绝对误差和(SAD)

$$SAD(i,j)=\sum_{m=0}^{M}\sum_{n=0}^{N}\left|f_k(m,n)-f_{k-\Delta t}(m+i,n+j)\right| \tag{4.3}$$

3) 均方误差(MSE)

$$MSE(i,j)=\frac{1}{MN}\sum_{m=0}^{M}\sum_{n=0}^{N}\left|f_k(m,n)-f_{k-\Delta t}(m+i,n+j)\right|^2 \tag{4.4}$$

4) 归一化互相关函数(NCCF)

$$NCCF(i,j)=\frac{\sum_{m=0}^{M}\sum_{n=0}^{N}f_k(m,n)f_{k-\Delta t}(m+i,n+j)}{\left[\sum_{m=0}^{M}\sum_{n=0}^{N}f_k{}^2(m,n)\right]^{1/2}\left[\sum_{m=0}^{M}\sum_{n=0}^{N}f_{k-\Delta t}{}^2(m+i,n+j)\right]^{1/2}} \quad (4.5)$$

fk，fk-△t 分别是当前帧和参考帧的像素值，MAD、MSE、SAD 等准则中以取最小点为最优匹配点，NCCF 准则中取最大点为最优匹配点。由于 SAD 准则不需做乘法运算，实现简单方便，所以运动估计算法多采用 SAD 为匹配代价函数。

4.4.2 全搜索算法 FS（Full Search）

全搜索算法[31]也被称为穷尽搜索法，是一种搜索策略最简单的搜索算法，它是对搜索范围内的所有可能位置计算 SAD(i，j)值，从中找出最小绝对误差和 SAD，其对应的位移量即为所求的运动矢量。由于对所有位置都进行搜索，搜索的结果是非常精确的，但计算量也是非常大的。下图 4.10 为全搜索示意图：

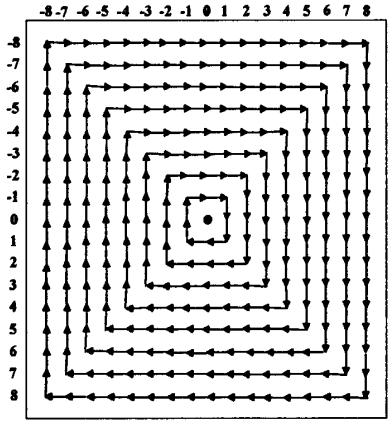


图4.10 全搜索示意图

FS 算法描述：

1) 从搜索窗中心开始，按顺时针方向由近及远地在逐个像素点处计算 SAD 值，直到遍历搜索范围内的所有点为止；

2) 在所有的 SAD 中找到最小误差 MBD(Minimum Block Distortion)值点，该点所在的位置即对应最佳运动矢量。

算法特点：

FS 算法是最简单、最原始的块匹配算法，由于遍历全局，因此找到的匹配点一定是全局最佳的，可靠度高，通常是其他算法性能比较的标准，但是它的计算量非常大，搜索时间需要很长，因此限制了它在实时压缩场合的应用。

4.4.3 钻石搜索算法（菱形搜索算法）

菱形搜索法(Diamond Search，也称钻石搜索法)最早是由 Shan Zhu 和 Kai-Kuang Ma[32]提出的，后又经过多次改进，目前已成为快速块匹配运动估计中性能优异的算法

之一。

DS 中采用了两种搜索模板，一种是大钻石模板(Large Diamond Search Pattern，LDSP)，另一种是的小钻石模板(Small Diamond Search Pattern，SDSP)，如图 4.11 所示。

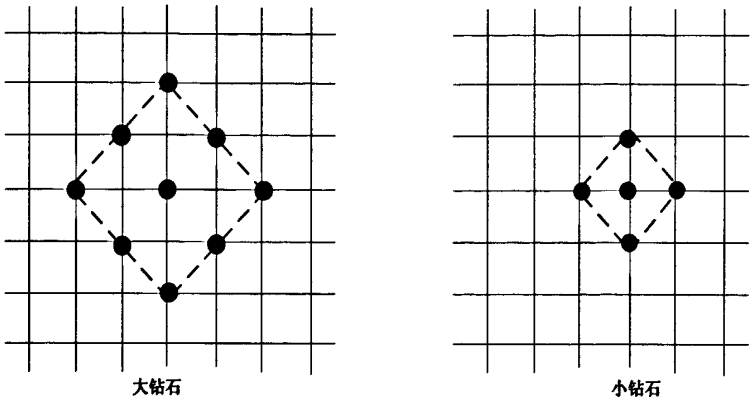


图4.11 钻石搜索法

DS 的算法流程如下：

1) 初始化大菱形LDSP以为搜索窗口的原点(0,0)为中心，测试LDSP的九个检测点。如果最佳匹配点为 LDSP 的中心，跳到第三步；否则进入下一步。

2) 移动 LDSP 中心到上一步中所得的最佳匹配点处，重新构造一个新的 LDSP，如果新的最佳匹配点位于 LDSP 的中心，进入下一步；否则重复这一步。直到检测点到达搜索窗边缘，终止搜索。

3) 切换搜索模板为 SDSP，检测其外围四点，从而得到目标运动矢量。

由于钻石搜索算法不限制搜索的步数，既对各个方向都进行搜索，又重点考虑水平竖直方向，所以可使搜索避免陷入局部最优，从而得到较好的性能。

4.4.4 六边形搜索算法

六边形算法与 DS 算法类似，采用两种搜索模板，六边形大模板的参考点均匀分布在以中心点为圆心的圆上，可以避免 DS 算法中常常出现的距离不一致的问题，六边形的小模板退化为 DS 小模板。由于大六边形模式相对于其它模式来说更接近于以 2 为半径的圆，所以搜索效率更高。六边形搜索算法如图 4.12 所示，采用大六边形模式和小六边形模式，小六边形模式实际上就是小菱形模式(CSDSP)。HEXBS[33]首先以搜索窗的中心点(0，0)为中心，采用大六边形模式进行搜索，如果最优点不是中心点，那么以最优点为中心继续进行大六边形模式搜索[34]，否则切换到小六边形模式(CSDSP)[35]，搜索小六边形的 4 个点，输出最终结果。

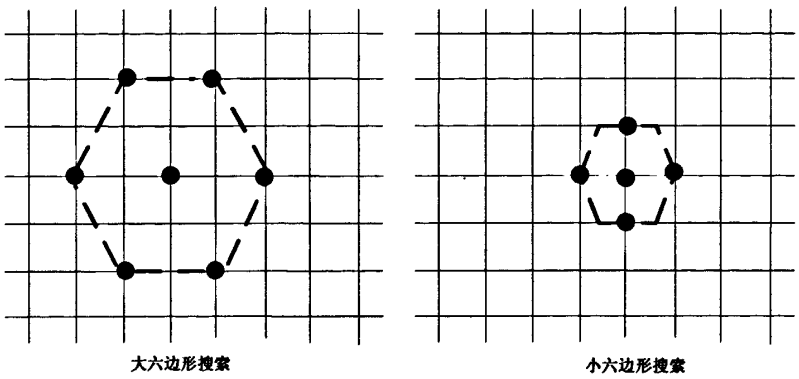


图4.12 六边形搜索法

4.5 非对称十字型多层次六边形格点搜索算法（UMHexagons）及其模板选择的优化

4.5.1 UMHexagons 算法

目前 H.264 推荐的算法是非对称十字形多层次六边形格点搜索算法（UMHexagons），这种算法用多种组合框架来代替单一框架，有效的提高了预测的有效性和健壮性。虽然，UMHexagons 算法搜索的步骤较多，但是由于不同框架的应用，网络覆盖的提高，提前中止条件的应用，此算法在预测效果上非常接近于全搜索算法[36]。

UMHexagonS[34,35]全名叫“非对称十字型多层次六边形格点搜索算法”(Unsymmetrical-Cross Muti-Hexagon-grid Search)，是一种整像素的快速运动估计算法。这是一项有中国专利的算法，其搜索过程比较复杂，采用了非对称交叉搜索(Unsymmetrical-crosssearch)，不均匀多六边形格点搜索(UnevenMulti-Hexagon-grid Search)，扩展六边形搜索(Extended Hexagon-based Search)等多种先进的方法[37]，是一种混合的搜索方式。图 4.13 展示了 UMHexagonS 搜索算法的主要搜索步骤。

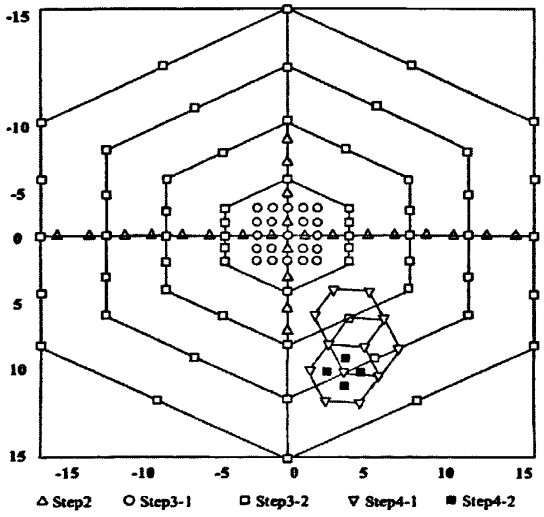


图4.13 UMHexagons算法的搜索过程

非对称十字型多层次六边形格点搜索算法（UMHexagonS）采用分等级搜索策略，称其为混合搜索算法是因为它包括四个主要的搜索过程： (1)初始搜索点的预测 ；(2)非对称的十字型搜索；(3)非均匀多层六边形格点搜索；(4)基于扩展的六边形搜索。

1) 初始搜索点的预测[36,37]：这是依据待估计块与周边块有很强的相关性，利用周边已编码的块进行初始点预测，预测模式有四种：

第一种：中值预测：P 为当前待估计块，与 A、B、C、D 的位置关系如图 4.14 所示。利用空间相关性，取 A、B、C 的运动矢量的中值，P 的预测矢量为 MV_{pred}：

$$\overline{MV_{pred}} = median(\overline{MV\_A}, \overline{MV\_B}, \overline{MV\_C}) \tag{4.6}$$

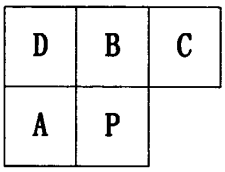


图 4.14 中值预测

预测矢量要遵循下面的规则：如果 A 处于图像或者 GOB 边缘的外面，用(0，0)块来代替；当 C 处于图像或者 GOB 边缘的外面，用第三块来代替；当 B 和 C 处于图像或者 GOB 边缘的外面，用 D 来代替。

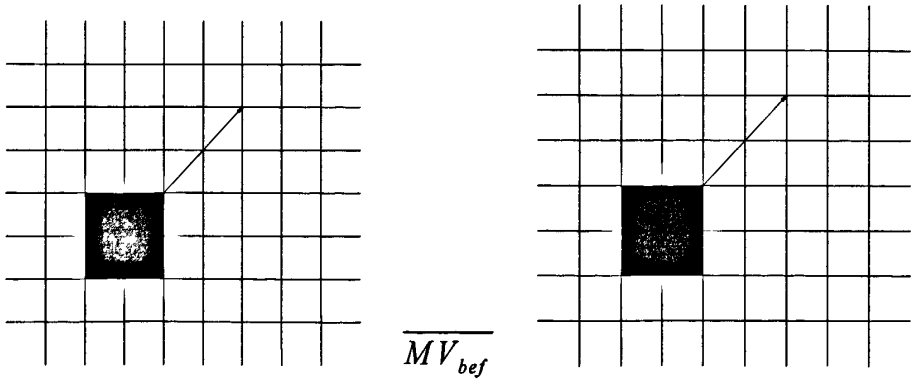
第二种：上层预测

利用 H.264 运动估计多宏块划分等特点，采用从模式 1 到模式 7 的分级搜索顺序，取已求出的、同位置、上一级、大一倍块的运动矢量；

第三种：相应块的预测

一般情况下，物体的运动轨迹是连续的，在时域中运动矢量存在很强的相关性，利用最近一帧相应块的运动矢量作为待预测块的运动矢量预测，如下图 4.15 所示：

$$\overline{MV_{pred}} = \overline{MV_{bef}} \tag{4.7}$$



(a) 帧 t-1　　(b) 帧 t

图4.15 相应块预测

第四种：时间域的相邻参考帧预测

H.264 采用多参考帧补偿来增加预测精度和编码效率，不同参考帧中的运动矢量有很强的相关性；利用其时间相关性，取已求出的，前一参考帧中当前块的 MV 按比例进行预测。

2) 非对称十字型搜索[38]

自然界图像序列在水平方向的运动比垂直方向的运动更加普遍[39]，因此非对称十字型搜索可以比较精确的预测到最佳的运动矢量，使用一个水平搜索范围为 W 和垂直搜索范围为 W/2 的非对称十字型搜索模式，这可以看成是一个简单而有效的预测模式来为下一步搜索获得一个精确的起始搜索点[40]。对于大范围运动的序列，可以将垂直范围扩大到 W。

3) 非均匀多层六边形格点搜索

该步搜索分成两小步：首先，以目前最佳点为中心，在范围为-2 到 2 的方形区域进行小矩形窗搜索。然后进行多层次六边形格点搜索，16 点的多层次六边形格点搜索通过伸缩系数对六边形进行扩展，在每一层次中都要判断是否要提前截止。16 点的多层次六边形格点搜索可以更好的处理大范围和不规则的运动，避免过早的落入局部最优。

4) 扩展的六边形搜索

这一过程也有两个步骤：首先以搜索中心进行大六边形搜索，如最佳点不在六边形中心则重复本步骤，如果在六边形中心则进入下一子步骤；然后以上一子步骤得到的最佳点进行小六边形搜索，搜索的结果作为最后的匹配点。

4.5.2 模板选择的优化

1）上面运动估计方式均没有考虑 H.264/AVC 的其他特征，仍然只是一个单纯的运动估计算法。他们相对于其他传统的搜索方法在速度和效果上都有较大的改善，但是它们对图像序列中存在的相关性仍然没有充分挖掘，如果先根据图像的相关性对搜索块的运动范围提前加以判断并进行更有针对性的搜索则可以减少进一步计算冗余，提高搜索效率。

在搜索起点方面，就是预测矢量的选择上，主要有时域和频域两大类，考虑到空域预测矢量的相关性比时域预测矢量强，并且因为时域预测需要存储帧间参考块的运动矢量，增加了额外的内存开销，对于较大尺寸的图像尤其庞大，不利于硬件实现，则空域预测矢量是最佳的选择;又考虑到 U pLayerM V 在 H. 264/AVC 中效果比较明显。因此在本文中只选择中值预测矢量，(0, 0)和上层模式预测矢量来进行预测。

在搜索路径方面，也就是搜索模板，根据运动上下文采取合适的搜索模板可以有效的提高搜索效率。根据运动上下文信息有针对性的采用 4 种搜索模版，它们的示意图如图 4.16 所示。它们分别定义如下:

菱形搜索模板:

{(X,Y)|(-1,0),(1,0),(0,-1),(0,1)};

小六边形搜索模板:

{(X,Y)|(-2,0),(2,0),(-1,-1), (1, 1) , (-1, 1) , (1,-1)};

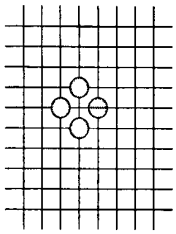
大六边形搜索模板:

{(X,Y)|(-2,0),(2,0),(-1,-2), (1, 2) , (-1, 2) , (1,-2)};

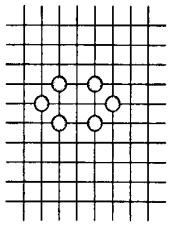
样点系搜索模板:

{(X,Y)|(2*i-1,0),(0,2*i-1),(1-2*i,0), (0,1-2*i) , (2, 2) , (2,-2) , (-2, 2) , (-2, -2),i=1,2,…, N };

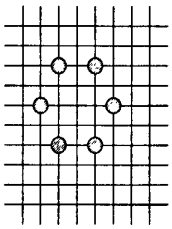
其中 N 为当前最大搜索范围。



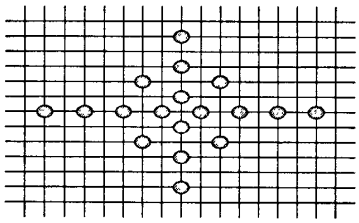
(a)菱形搜索模板



(b)小六边形搜索模板



(c)大六边形搜索模板



(d) 样点系搜索模板

图 4.16 4 种搜索模板图

由模板的形状可知,菱形模板适合于运动搜索块运动范围较小的情况;大小六边形搜索模板适合于运动范围适中的情况;样点系模板搜索适合于运动范围较大并且较复杂的情况。

结合图 4.17 具体介绍本文快速算法。本算法先按照搜索块运动量的大小将块大致分成静止或微小运动块，中等运动块和大运动块 3 种类型，每种类型块区别对待。由于实际视频序列中处于静止区域较多，且相邻块的运动存在相关性，故首先计算可能性较大的预测矢量点和零矢量点。若此时 J(m, λ_{motion})最小点为零矢量点，则表示本块为静止块或者微量运动块的几率最大，则以菱形搜索一次检测出最佳运动矢量;对于中等运动块，结合搜索范围的限制和当前搜索信息，通过大小六边形搜索和菱形搜索相结合来检测出最佳运动矢量;对于大运动块，则先结合块本身的尺寸先确定出粗略运动范围，然后再参照与中等运动块相似的搜索方式来进行精细搜索以确定出最佳运动矢量，其中，粗略运动范围对于 16 X 16 模式块由样点系搜索来确定的，对于非 16 X 16 模式块则是通

过参考本块的上层模式运动矢量点来确定的。

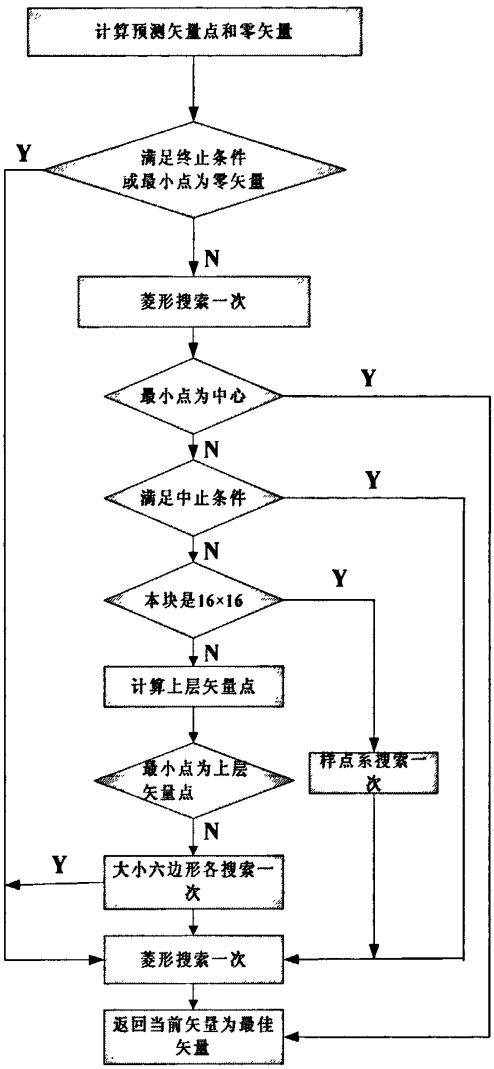


图 4.17 快速运动估计算法流程图

2）实验在 Intel Pentium4 CPU2.8G，内存大小为 DDR1G 的环境下进行，采用 JVT 提供的参考软件 JM 10. 1 为实验平台，程序配置如下:编码帧结构为 IPPP. . .，编码帧总数为 60，相邻 I 帧间隔为 30 帧，参考帧数 5，搜索范围为(-16, 16)，量化值为 28;不使用码率控制和率失真优化。实验用 Claire. qcif(图像变化缓慢，细节不丰富)，Carphone. Qcif, Foreman.qcif(图像变化比较剧烈，细节较丰富)，Trevor. qcif(图像变化较剧烈，细节较丰富)4 个典型序列。实验数据统计见表 4.2。

表中 FS(全搜索法)，UMHexagonS 算法和 Diamond Search 算法都是 JVT 标准参考软件 JM10.1 给出的算法，Proposal 为本文给出的算法;PSNR 增加量是指本文算法分别与 JM10.1 中几种算法的 PSNR 的差值;编码时间的相对值是指各种快速算法相对于全搜索算法的相对编码时间单位，在此设定全搜索编码时间为 100 个时间单位。编码时间节

省百分比是指本文算法分别相对于 JM10.1 中几种算法的全部编码时间节省的百分比。

表4.2 QCIF序列测试结果

Sequence	Algorithm	PSNR (dB)	A(dB)	T_{code}	B/%
Carphone (qcif)	FS	37.40	-0.05	100	-83.6%
	UMHexagonS	37.35	0	25.5	-35.7%
	DS	37.36	-0.01	19.4	-15.5%
	Proposed	37.35		16.4	
Claire (qcif)	FS	39.94	-0.02	100	-90.9%
	UMHexagonS	39.91	0.01	20.9	-56.5%
	DS	39.93	-0.01	14.2	-35.9%
	Proposed	39.92		9.1	
Claire (qcif)	FS	35.87	-0.01	100	-84.4%
	UMHexagonS	35.90	-0.03	27.8	-43.9%
	DS	35.87	-0.01	20.2	-22.8%
	Proposed	35.86		15.6	
Trevor (qcif)	FS	36.14	-0.03	100	-83.1%
	UMHexagonS	36.12	-0.01	28.1	-39.9%
	DS	36.07	-0.04	22.2	-23.9%
	Proposed	36.11		16.9	

（1）对这几个视频序列，在编码质量方面，本算法 PSNR 比全搜索算法平均下降为 0.028 dB,减少最大量为 0.05 dB，对于活动图像的接受者人而言，这点差别主观上是不易察觉到的;在编码速度方面，本算法对各种运动类型的视频序列都有较大的提高，它比 FS 算法，U M H exagonS 算法和 Diamond Search 算法分别平均节省 85.5%，44.2%和 32.1%的编码时间。

（2）本算法对其他几种算法的改进程度与实际序列的运动特点有关。序列细节的丰富程度和图像变化的剧烈程度对编码速度有较大的影响，但是由表中的数据统计可知，本文算法相对于各类序列的编码速度都有较大提高。

(3)对 Foreman.qcif 序列,UMHexagonS 算法的 PSNR 比全搜索算法的 PSNR 还要高出 0. 03 dB，这是因为几种搜索方式采用的匹配准则都是 J(m, λ_{motion})，它是图像匹配度和编码速率的综合反映，而 PSNR 反映的只是图像匹配度的信息，故全搜索法的 J(m, λ_{motion})在综合性能上为最优而 PSNR 未必一定是最高的。

4.6 模式选择的改进的方向性菱形搜索

4.6.1 运动类型判定

记 P 帧当前编码第 k 个宏块亮度像素值为 xi，j（1≤i，j≤16），其参考帧对应位置参考宏块亮度像素值为 ci，j（1≤i，j≤16），则宏块与其参考宏块的 MAD 值为

$$MAD_k = \frac{1}{256}\sum_{i=1}^{16}\sum_{j=1}^{16}|x_{i,j}-c_{i,j}| \tag{4.8}$$

记每帧图像宏块总数为N，则所有宏块MAD值的均值为

$$\overline{MAD}=\frac{1}{N}\sum_{n=1}^{N}MAD_n \tag{4.9}$$

当宏块MAD值满足如下关系时，判定当前宏块处于背景区域

$$\frac{MAD_k}{\overline{MAD}}<T \tag{4.10}$$

其中T为判决门限值。对于当前帧所有N个宏块的MAD值，其方差表示如下

$$\sigma=\sqrt{\frac{1}{N}\sum_{k=1}^{N}(MAD_k-\overline{MAD})^2} \tag{4.11}$$

考虑到背景区域的判定与当前帧的量化参数有关，一般来说，量化参数越大，宏块残差经量化后高频信息越少，越倾向于判定为背景区域，因此综合考虑宏块的MAD值与其方差的背离程度及量化参数，经过实验，判决门限可由如下经验表达式确定

$$T=\sqrt{\frac{\sigma}{\overline{MAD}}}+a(QP-b) \tag{4.12}$$

式中参数a与b可通过经验确定，实验中分别取0.05和32。

4.6.2 运动估计算法描述-改进的方向性菱形搜索算法

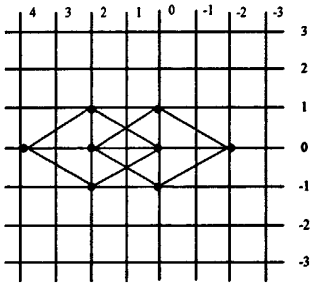


图4.18（a）未改进的方向性菱形水平搜索

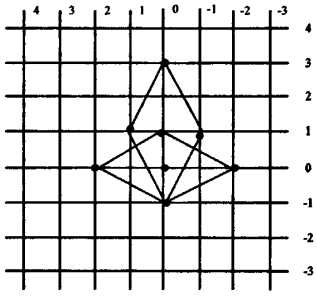


图4.18（b）未改进的方向性菱形垂直搜索

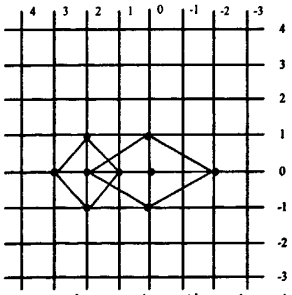


图4.18（c）改进的方向性菱形水平搜索

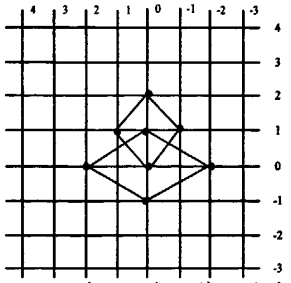


图4.18（d）改进的方向性菱形垂直搜索

改进的方向性菱形搜索的原理如图1所示，其中图4.18(a)，图4.18(b)是方向性菱形模板搜索策略描述，图4.18(c)，图4.18(d)是改进的方向性菱形模板搜索策略描述。从图4.19中可知，如SAD值最小点不是中心点时，采用未改进的方向性菱形搜索策略，使用水平方向或垂直方向菱形搜索模板会遗漏(3，0)和（0，3)点，造成搜索的不精确。改

进的搜索策略在 SAD 值最小点处使用小菱形模板搜索，非继续使用方向性菱形模板，就不会遗漏中心点旁的两点(2，0)和(0，3)。

通过设置阈值 T 判定宏块运动类型后，可以针对不同的宏块类型采用不同的模板，当宏块缓慢运动时，用小菱形模板及搜索策略保证了的匹配精度；宏块剧烈运动时，用方向性模板及搜索策略加快了搜索的速度，而且比通常方向性搜索具有更高的精度。因此算法的搜索点数比菱形算法少很多，与方向性菱形搜索相当，但搜索的精度比方向性菱形算法高。

4.6.3 带有模式选择的运动估计优化策略

第一步 首先方向性菱形模板的水平模板和垂直模板交替使用，对当前宏块进行模式检测，根据所得到的运动矢量判断下一步所进行的搜索模式。

第二步 检测 SKIP 模式，若其率失真代价函数值 J(Skip)满足 J(Skip)≤T1，则令 SKIP 模式为最优编码模式，并转入第七步；

第三步 检测 inter16×16 模式，若满足 J(Skip)-J(inter16×16) ≤T2，则从 SKIP 和 inter16×16 中确定最优编码模式并转入第七步；

第四步 若 J(inter16×16)<=T3，将帧内编码模式标志位置 0，为不可选；

第五步 若式（5）成立则检测 inter16×8 和 inter8×16 两种编码模式，最优运动矢量的搜索首先在整数像素上进行，然后继续计算其相邻八个半像素点的 SAD，进行分数像素匹配。为了考察当前编码块内运动纹理特征是否类似，计算水平方向左右两侧的半像素位置 SAD 的差值 H，以及垂直方向上下两侧的半像素位置 SAD 的差值 V，比较两者的大小。置 intra16×16 标志位为可选。

（1）如果 H>V，此宏块含有水平运动矢量，直接排除掉其他模式，只选用 16×8 模式，采用改进的方向性菱形搜索进行水平搜索，得到的最优点为全局最优点，结束其它模式的搜索，转到第七步；

（2）如果 V>H，此宏块含有垂直运动矢量，直接选用 8×16 搜索模式，使用改进的方向性菱形进行垂直搜索，得到的最优点为全局最优点，结束其它模式的搜索，转到第七步；

（3）如果 H 和 V 相等，则模式 16×8 模式 8×16 均进行运动矢量搜索，采用菱形搜索算法进行运动估计，转到第七步；

第六步 用菱形搜索算法检测剩下的模式，并置 intra4×4 为可选。

第七步 对当前宏块存在的编码模式进行率失真 RDO 比较，选择最优模式为最终编码模式。

由经验值发现静止宏块的平均 SAD 值介于 600～1300 之间。阈值越高，误判的

可能性就越大，因此保守地使阈值 T1 等于 512，T2 等于 160，使得在获得较大加速的同时，不会造成明显的视频质量下降，T3 为量化参数 QP 的函数[6]，如下所示:

T3=8(QP)2

（6）

整个算法的流程图如图 4.19 所示:

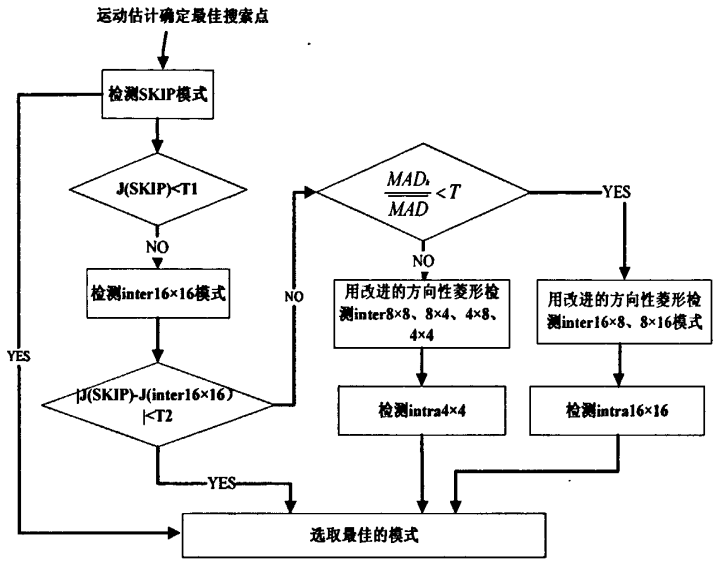


图 4.19 带有模式选择的改进的方向性菱形搜索算法流程图

4.6.4 仿真结果与性能分析

为验证文中算法，此次实验在开源代码 x264 编码参考基础上进行选取了几个典型的快速搜索算法为比较：菱形搜索算法 DS、六边形搜索 HEXBS 算法实验，同时为了标志模式选择的对算法的影响，将改进的方向性菱形算法和带模式选择的改进方向性菱形算法分开测试。选择了 4 个标准测试序列进行仿真实验，其序列为：Akiyo、container、foreman、carphone， QCIF 格式，一个参考帧，采用 Baseline 框架，无 B 帧，QP 取值为 26，测试结果如表 4.3 所示。

表4.3 QCIF序列测试结果

Sequence	性能指标	DS	HEXBS	改进的方向性菱形算法	带模式的改进的方向性菱形算法
akiyo	帧率 fps	314.24	318.31	323.45	416.02
	PSNR dB	38.639	38.636	38.633	38.630
	码率 kb/s	30.16	30.19	30.16	30.16
Container	帧率 fps	314.19	333.96	290.02	400.34
	PSNR dB	34.652	34.639	34.630	34.615
	码率 kb/s	31.51	31.48	31.47	31.47
Foreman	帧率 fps	265.30	269.50	273.28	347.08
	PSNR dB	28.712	28.698	28.670	28.657
	码率 kb/s	29.78	29.73	29.70	29.72
Carphone	帧率 fps	257.68	272.66	280.62	340.06
	PSNR dB	30.050	30.045	30.043	30.041
	码率 kb/s	32.71	32.71	32.71	32.71

表4.4算法性能对比

Sequence	ΔPSNR（dB）	提高的编码速率 %
akiyo	DS:-0.009	32
	HEXBS:-0.003	30
Container	DS:-0.037	27
	HEXBS:-0.024	20
Foreman	DS:-0.009	31
	HEXBS:-0.003	29
Carphone	DS:-0.009	32
	HEXBS:-0.004	25

从表 4.3 中可以看出，不管是剧烈运动还是缓慢运动序列，在性能方面，DS 的 PSNR 值最高，与 DS 相比 HEXBS 更快，因为 HEXBS 每步中都只增加三个检测点，而 DS 依据最佳匹配点出现位置分别增加五个或三个点。当搜索沿水平或垂直方向进行时，两者步长均 2，但 HEXBS 只需检测三个点，而 DS 是五个；当搜索沿对角线方向进行

时，两者的检测点数一致，均为3个，但步长不一致，HEXBS 为5 ，DS 为2。

从表4.3中可以看出，不管是剧烈运动还是缓慢运动序列，在性能方面，DS的PSNR值最高，与 DS 相比HEXBS更快，因为HEXBS 每步中都只增加三个检测点，而 DS依据最佳匹配点出现位置分别增加五个或三个点。当搜索沿水平或垂直方向进行时，两者步长均为 2，但 HEXBS 只需检测三个点，而 DS 是五个；当搜索沿对角线方向进行时，两者的检测点数一致，均为3个，但步长不一致，HEXBS为5 ，DS 为 2。

改进菱形搜索算法与HEXBS算法相比，速度较低一点，其原因是该算法在方向性菱形搜索的基础上做了改进，当搜索剧烈运动块时，若中心点的SAD值不是最小值时，则在远点或者近点处使用小菱形搜索，这样就可以避免因方向性水平或垂直搜索而遗漏一些点，搜索更全面，但是也因此会导致搜索时间略有增长。

而带模式的改进的搜索算法搜索速度最快，原因是采用了类型判断加运动估计搜索联合决策方法，在搜索进行之前就对模式进行了筛选，去除了不适合模式。但因有可能出现运动类型错判的影响，PSNR值也略有降低，从表4.4中可以看出PSNR下降不超过0.04dB，但速度却较之DS算法上编码速度上升了27～32%，与HEXBS编码速度上升了25～30%。从人眼视觉的生理特性来看，当质量下降在-0.5db以内，人的主观感觉并不会有太大的变化。所以，本算法基本上不会影响视频的主观效果。

4.7 帧内预测算法的优化

H.264/AVC 帧内预测编码利用了图像空域的冗余度，首先用上边和左边宏块(或者块)中同当前宏块(或者块)相邻的像素值预测当前宏块(或者块)，然后对该预测值同当前宏块的残差进行变换、量化和熵编码。对于亮度分量，帧内预测可以采用I4或者I16两种方式，色度分量为8×8的块。I4亮度模式的分为9种预测模式，X264的帧内编码流程如下图4.20所示：

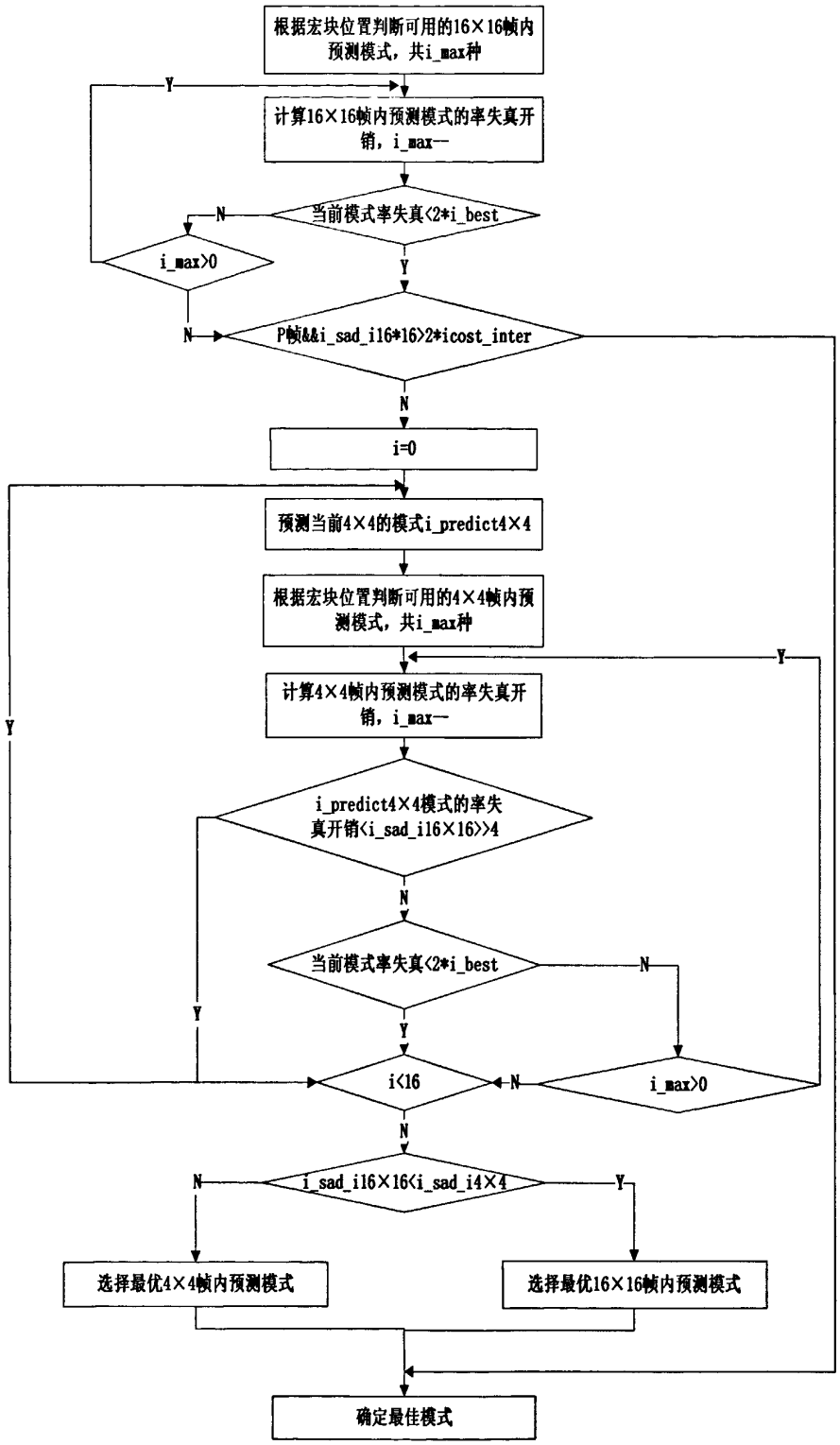


图4.20 X264的帧内编码流程

为了达到最佳的编码性能，编码器对所有的帧内预测模式做一遍，取 RDO 性能最好的一个。帧内预测一共有 M8×(M4×16+M16)=4×(9×16+4)=592 种模式，其中 M8、M4 和 M16 分别为 8×8 色度模式、I4 亮度模式和 I16 亮度模式。除了对 I16 模式采用了

SATD 算法外，其他的模式总共进行了 M8×(M4×16)=576 次的 RDO 计算，每次 RDO 都要做残差，变换，熵编码，计算量十分巨大。为了解决帧内预测模式选择计算复杂度高这个瓶颈，需要对帧内预测算法进行优化。

1) 充分挖掘 Intra_16×16 和 Intra_4×4 这 2 种预测方式之间的相关性，用待编码宏块在 Intra_16×16 方式下的 SAD 值来衡量宏块的平滑程度，从而判定是否可以不执行 Intra_4×4 方式的预测过程。

2) 根据 SATD 特征对 Intra_4×4 方式的预测模式进行筛选，选择出可能性大的模式，排除掉可能性小的模式，避免不必要的 RDCost 的计算，从而降低计算复杂度。

图 4.5 显示了各个模式的预测图样，除去 DC 预测模式外，其它的都是方向预测模式。块内同一方向上的像素点具有相同的预测值，以此来近似逼近不同方向纹理特性的图像。

根据 4×4 亮度的 9 种预测方向，改进的帧内编码流程如下图 4.21 所示：

1) 确定 Intra16×16 方式下的最优预测模式，步骤同 H.264/AVC 原算法；

2) 计算宏块在 Intra16×16 方式下的率失真代价 RDCosti_16×16。算法同 H.264 原算法同；

3) 计算宏块的 SAD。如果 SAD<T1(QP<=20 时，T1=500；QP>20 时，T1=1000)，则跳到步骤(6)；

4) 确定 Intra4×4 方式下的最优预测模式，将待编码宏块分为 16 个 4×4 块，分别确定每个 4×4 块的最优预测模式；

(1) 计算当前 4×4 块的最有可能模式 MPM 及 9 种预测模式的 SAD；

(2) 如果 MPM 为模式 0，3，7 或 5，6 中的一种，则选择相近模式作为计算 RD_cost 的候选模式；

(3) 如果 MPM 为模式 2，4，8 中的一种，则计算这 9 中预测模式的 SATD 值的平均值 Value，将 SATD 值小于 Value 的预测模式作为计算该模块 RD_cost 的候选模式；

(4) 分别对每种候选模式对应的预测残差块进行 DCT 变换/量化、反 DCT 变换/反量化，并计算其编码代价 RD_Cost；

(5) 比较所有候选模式的 RD_cost，选择具有最小 RD_Cost 的预测模式为当前 4×4 块的最优预测模式；

5) 结合色度编码，统计待编码宏块在 Intra_4×4 方式下的率失真代价 RDCosti4×4；

6) 确定最优的帧内预测模式，步骤同 H.264/AVC 原算法。

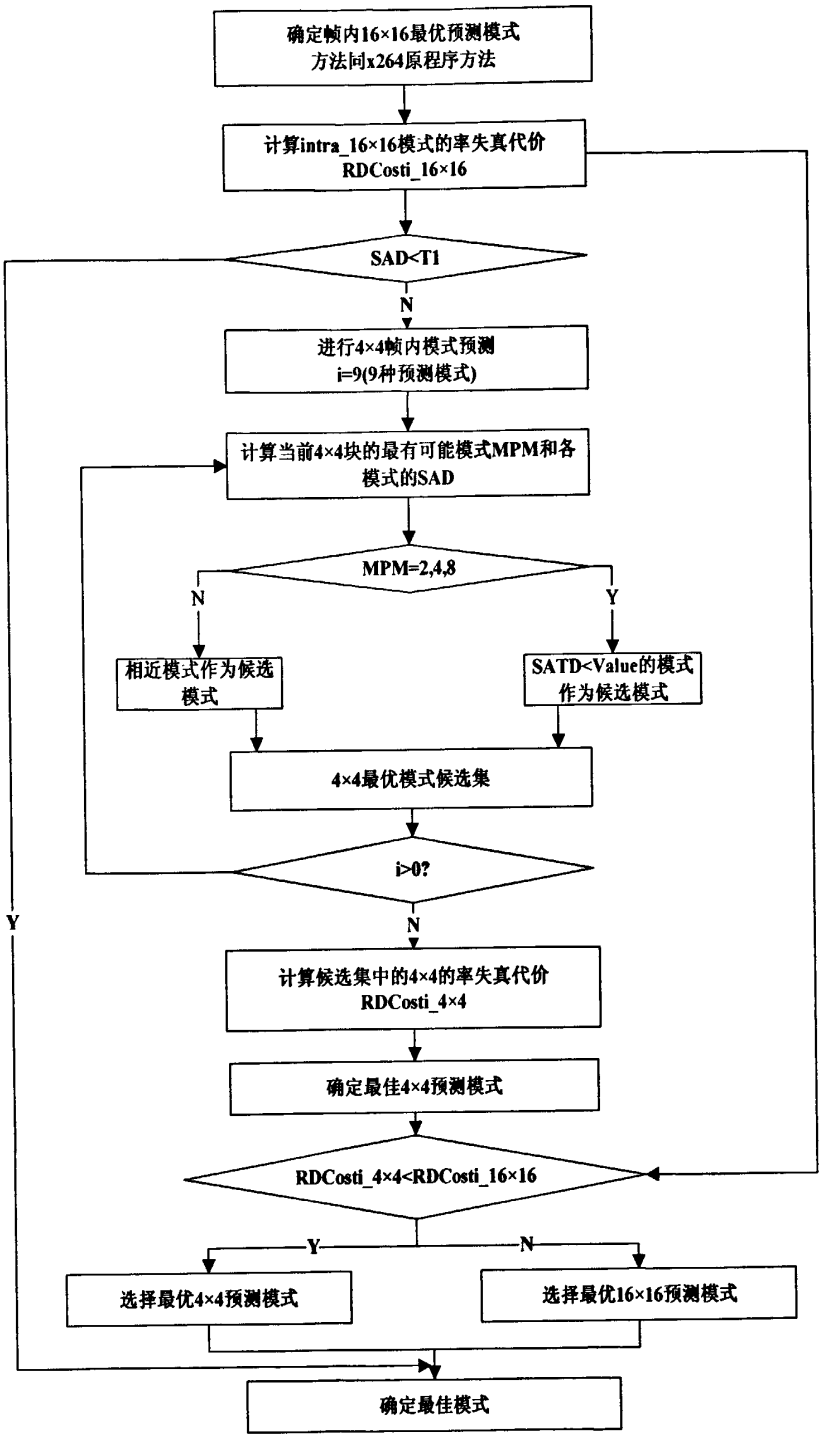


图4.21 改进的帧内编码流程

4.8 本章小结

本章以X264作为参考模型详细分析了H.264/AVC算法中各个模块所采用的技术细节，包括帧内预测，帧间预测，整数变换和量化以及熵编码等各个模块所采用的新技术。对各个模块的算法复杂度进行了分析，并确定了优化工作的重点模块，对核心的运动估计算法和帧内预测算法进行了改进和优化，降低了编码器的算法复杂度。

第五章 视频采集编码模块设计

本系统运行的平台是基于 S3C6410 的嵌入式 Linux 系统。系统在启动后，启用了 MMU，系统进入保护模式，这样应用程序就不能直接读写外设的 I/O 区域，这时一般要借助于该外设的驱动来进入内核完成这个工作。所以系统中的视频采集分为两步实现：一是为 USB 接口的数码摄像头在内核中写入驱动，二是编写上层应用程序获取视频数据。

5.1 V4L2 视频采集程序设计

监测终端使用的是拥有 USB 接口的外置数码摄像头，USB 设备和嵌入式硬件开发平台构成的一个 USB 通信系统。Linux USB 主机驱动由三部分组成[41]。主机控制器驱动程序(HCD)、USB 驱动(USBD)及设备端驱动程序(Slave Device Driver)，相互关系如图 5.1 所示。

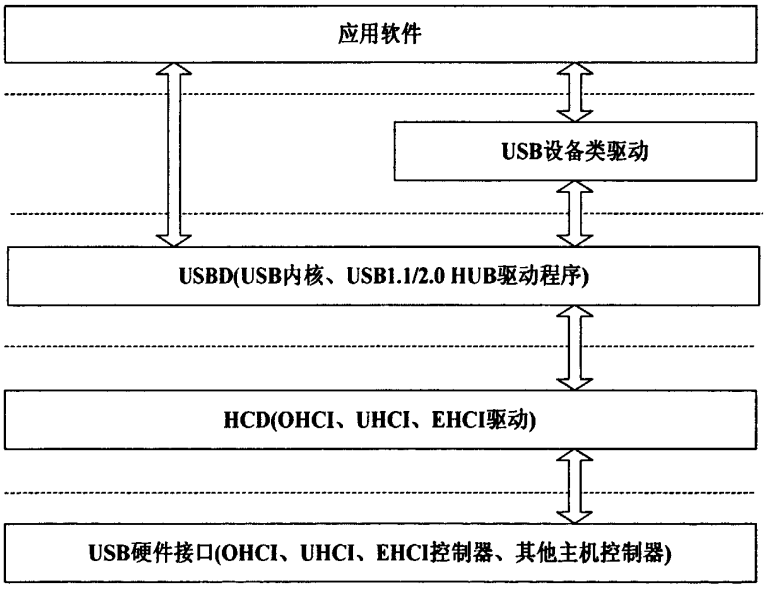


图5.1 Linux USB驱动程序结构

HCD 是 USB 主机驱动程序中直接与硬件交互的软件模块；USBD 部分是整个 USB 主机驱动的核心，提供 USB 总线管理，带宽管理等；USB 设备端驱动是最终与应用程序交互的软件模块，向应用程序屏蔽了硬件实现的细节，使得应用程序可以像操作普通文件一样来操作外部设备，即可以使用和操作文件中相同的、标准的系统调用接口函数来完成对硬件设备的打开、关闭、读写和 I/O 控制操作。为应用程序提供访问接口。

对于 USB 摄像头来说，其核心是感光芯片和数据处理 DSP 芯片。其中 DSP 芯片是

影响摄像头视频采集速度以及相关性能的主要因素。本系统选用的是中星微 ZC301，用它将摄取的数字视频图像直接通过 USB 接口送入开发板进行处理。

这款芯片的特点是内含数字摄像 IC 接口，DRAM 接口、实时图像压缩引擎、USB 接口、FIFO 等功能，通过采用影像光源自动增益补强技术，自动亮度、白平衡控制技术，色饱和度、对比度、边缘增强以及伽马矫正等先进的影像控制技术，搭配 COMS 感光芯片使其各项技术指标都能与 CCD 芯片相媲美，因此，可以满足视频监控的需要。

在 Linux 下，对驱动程序的编译添加一般有两种方式[42]。可以静态编译进内核，再运行新的内核来测试；也可以编译成模块在运行时加载。第一种方法效率较低，但在某些场合是唯一的方法。模块方式调试效率很高，它使用 insmod 工具将编译的模块直接插入内核，如果出现故障，可以使用 rmmod 从内核中卸载模块。不需要重新启动内核。但嵌入式系统是针对具体应用的，所以本系统将 Linux 下的 ZC0301 驱动程序通过 make menuconfig 配置以及内核重编译将设备驱动程序以静态的方法编译进内核，再将带有 ZC301 驱动的内核，烧写到开发板上即可使用。

视频采集相关的 Linux 内核配置主要是摄像头的 USB 驱动支持和 V4L2 支持。包括下列选项：

在 Device Drivers-> Multimedia devices 菜单项中，选中

<*>Video For Linux Two-><*>Enable video for linux API 1

<*>Video For Linux Two-><*>Enable video for linux API 1 compatible layer

Video Capture Adapters->V4L2 USB devices-><*>USB ZC0301[P] Image Processor and Control Chip support(NEW)

Linux 下的 Video for Linux Two，简称 V4L2。它为市场现在常见的电视采集卡和并口及 USB 口的摄像头提供统一的编程接口[43]。同时也提供无线电通信、文字电视广播解码和垂直消隐的数据接口。本系统采用的就是基于 V4L2 的。Linux 下与 V4L2 相关的设备文件与其用途如表 5.1 所示。

表5.1 V4L2的设备文件与用途

/dev/video	视频捕捉接口
/dev/radio	AM/FM 音频设备
/dev/vtx	文字电视广播
/dev/vbi	原始 VBI 数据

USB 摄像头在 Linux 系统中属于字符设备，其主设备号是 81。在嵌入式平台上使用 USB 摄像头时，需要先用 mknod 命令创建一个设备结点/dev/video0，再通过 ln 命令建立该结点与/dev/v4l2/video0 的链接。访问时，用 open()打开设备/dev/video0 即可。

利用 V4L2 对 USB 摄像头的编程需要 Linux 下两个系统调用，分别是 ioctl()调用和 mmap()调用。

ioctl 系统调用的功能是通过打开的文件描述符对各种文件尤其是字符设备文件进行控制，完成特定的 I/O 操作。V4L2 支持的 ioctl 命令在应用中主要用的如下表 5.2 所示：

表5.2 V4L2的主要控制命令

函数	功能
ioctl（fd，VIDIOCGCAP，& capability）	获取采集设备的基本功能信息
ioctl（fd，VIDIOCSPICT，& picture）	设置和获取采集图象的各种属性
ioctl（fd，VIDIOCGPICT，& picture）	
ioctl（fd，VIDIOCGMBUF，*mbuf）	获取缓冲区信息
ioctl（fd，VIDIOCMCAPTURE，&mmap）	捕捉图像，获取图像信息
ioctl（fd，VIDIOCSYNC，&frame）	等待捕获完改帧图像

mmap 调用的功能是实现内存映射，即将指定文件或对象的一部分映射到内存中去。这样可以通过访问特定的内存区域来直接对文件或对象进行存取。与 read 和 write 调用相比，这种 I/O 方式效率更高。

mmap 调用同样适用于设备文件，即可以用 mmap 调用将设备文件映射到内存中去，对设备文件的读写就转化为对内存的读写。对持续采集大量图像数据的摄像头来说，用 mmap 的内存映射方式来传送数据，更能体现效率。

mmap 调用的格式是：

void* mmap(void* start,size_t length,int prot,int flags,int fd,off_t offset);

参数 fd 是被映射到内存文件的描述符；start 指向对应的内存起始地址；length 为映射到内存的文件大小；offset 为目标文件中被映射部分起始点距文件开头的偏移量；mmap 调用成功后，其返回值就是指向内存映射区域的指针，该内存区域的大小就是由 length 参数指定的字节长度，接着应用程序即可对该区域进行存取操作。

V4L2 视频程序设计时必须声明两个头文件 sys/types.h 和 linux/videodev.h。然后，遵循 V4L2 给出 video_device 数据结构的定义：

```
typedef struct v4l2_input{
        int fd;
        struct video_capability capability;          /*设备的基本信息*/
        struct video_picture picture;                /*设备采集的图象的各种属性*/
        struct video_mmap mmap;                      /*用于设置 mmap*/
        struct video_mbuf mbuf;                      /*利用 mmap 进行映射的帧的信息*/
        unsigned char *map;                          /*存放返回地址*/
        char device[256];
        int width;
        int height;
        struct frame *f;
        pthread_t thread;
```

```
    int cur_frame;
    int framestat[2];                                   /*双缓冲*/
} video_device;
```

主要用到的采集操作函数定义如下：

```
int v4l2_open(char *name，video_device *vd);                    /*打开设备文件*/
int v4l2_close(video_device *vd);                               /*关闭设备文件*/
int v4l2_get_capability(video_device *vd);                      /*取得设备文件的相关信息*/
int v4l2_get_picture(video_device *vd);                         /*取得输入到摄像头的影像信息*/
int v4l2_set_palette(v4ldevice *vd, int palette);               /*设置采集图片格式*/
int v4l2_grab_init(v4l_device *vd，int width，int height);      /*采集初始化*/
int v4l2_grab_frame(video_device *vd，int frame);               /*将影像放到 mmap()映射的内存*/
int v4l2_grab_sync(v4l_device *vd);                             /*同步，等待一帧影像采集的完成*/
int v4l2_mmap(v4l_device *vd);                                  /*内存映射*/
int v4l2_get_mbuf(v4l_device *vd);                              /*得到用于内存映射的缓冲区信息*/
unsigned char *v4l2_get_address(v4l_device *vd);                /*将采集数据的起始地址以指针返回*/
```

视频采集流程图如下图 5.2 所示：

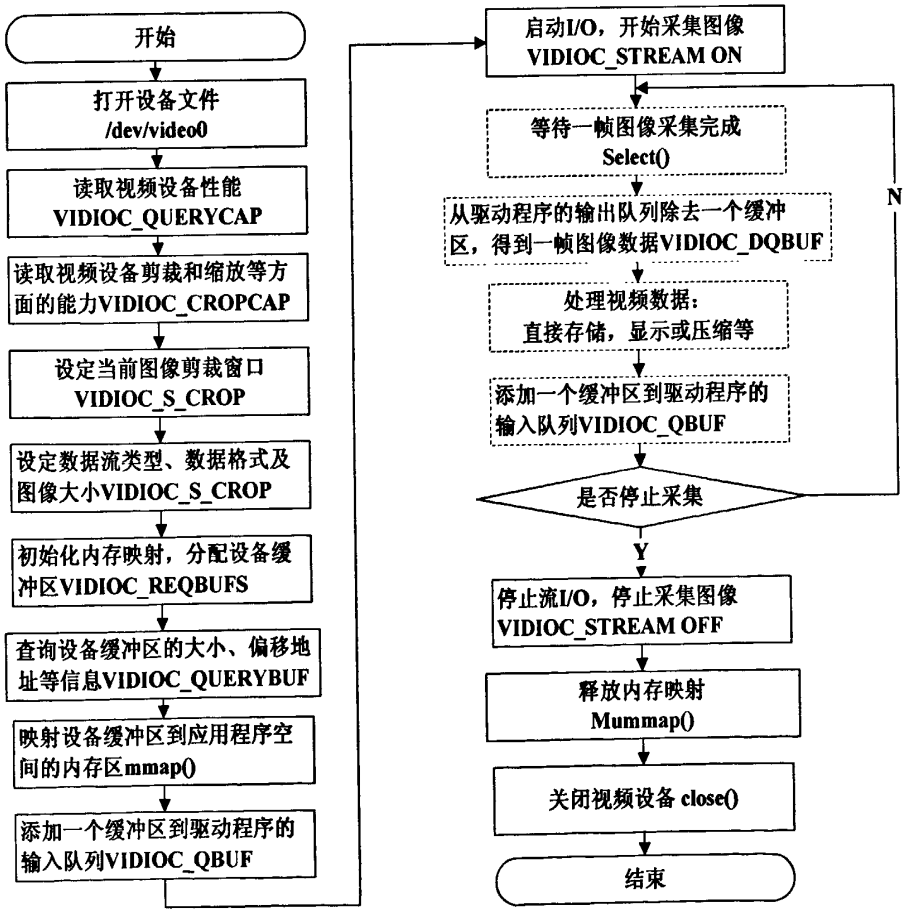


图5.2 V4L2视频采集流程图

5.2 视频采集压缩程序的实现

设计的视频采集与压缩程序完成视频原始数据的采集，H.264 标准的压缩以及 SD 卡存储功能。其中，视频采集基于上一节所实现的程序；视频压缩主要是基于开源工程 X264。由于 X264 是开放源代码的，将把上述两个程序合二为一，融合成为一个完整的视频采集与压缩软件。

使用 X264 程序的命令格式如下：

x264 默认选项 -o 输出文件 输入文件 [长×宽]

X264 的输入、输出都是文件形式。在本视频采集与压缩程序设计中，将修改 X264 程序的输入方式，使它直接从内存获得视频数据源。程序首先从 USB 端口采集摄像头的视频原始数据，存放在系统内存中，然后由 X264 进行压缩，以文件的形式输出，保存在根文件系统中。采集视频的分辨率为 QCIF。

根据上述设计思想，视频采集与压缩程序基于 X264 项目的源代码进行修改。通过分析 X264 项目的源代码发现，其项目主文件是 X264 源码顶层目录下的 x264.c 文件。x264.c 首先设定编码的默认参数；然后对命令行指定的部分参数进行解析，覆盖默认参数；最后调用编码核心算法，对读取的输入文件编码。

由于 X264 的输入是文件形式，不指定输入文件就无法运行，因此首先要在参数解析的过程中屏蔽掉它对输入文件的检测；其次，要在编码过程开始前增加 USB 摄像头视频采集的初始化程序；再次，在编码过程中增加视频采集的代码，以获取原始视频数据，代替原来的从文件获得原始视频数据。最后，在达到设定的编码帧数后，关闭视频采集。

下面列出修改、添加后的主要部分代码，x264.c 中的 Encode 函数修改后的代码如下，其中有部分文字说明：

```
static int   Encode( x264_param_t *param, cli_opt_t *opt )
{
    x264_t *h;
    x264_picture_t pic;
    int      i_frame, i_frame_total;
    int64_t i_start, i_end;
    int64_t i_file;
    int      i_frame_size;
    int      i_progress;
//****************** JCM ******************************
/*     i_frame_total = p_get_frame_total( opt->hin );
    i_frame_total -= opt->i_seek;
```

```
    if( ( i_frame_total == 0 || param->i_frame_total < i_frame_total )
    && param->i_frame_total > 0 )
    i_frame_total = param->i_frame_total;*/
//上面被注释掉的5行，原来是读取输入文件，得出其中的图像帧数，现在已经不需要，
//由下面一行代码设定编码帧数。
    i_frame_total = FRAMES;
//****************** JCM *******************************
    param->i_frame_total = i_frame_total;
    if( ( h = x264_encoder_open( param ) ) == NULL )
    {
        fprintf( stderr, "x264 [error]: x264_encoder_open failed\n" );
//****************** JCM *******************************
        //p_close_infile( opt->hin );
//****************** JCM *******************************
        p_close_outfile( opt->hout );
        return -1;
    }
    if( p_set_outfile_param( opt->hout, param ) )
    {
        fprintf( stderr, "x264 [error]: can't set outfile param\n" );
//****************** JCM *******************************
       //p_close_infile( opt->hin );
//****************** JCM *******************************
        p_close_outfile( opt->hout );
        return -1;
    }
//*************************** JCM add ******************************
    int width = param->i_width, height = param->i_height;
    int frame;
    int Ysize = width * height;
    int Usize = Ysize / 4;
    int Vsize = Usize;
    init_videoIn( &width, &height);
//上一行 init_videoIn 函数完成视频采集的初始化工作，并开始捕捉第一帧图像。是新增加的
    fprintf (stderr, "input palette: YUV420\n");
    fprintf (stderr, "size: %dx%d\n", width, height);
//*************************** JCM add ******************************
    /* Create a new pic */
    x264_picture_alloc( &pic, X264_CSP_I420, param->i_width, param->i_height );
    i_start = x264_mdate();
    /* Encode frames */
    for( i_frame = 0, i_file = 0, i_progress = 0;
         b_ctrl_c == 0 && (i_frame < i_frame_total || i_frame_total == 0); )
    {
```

```
//****************** JCM ******************************
//          if( p_read_frame( &pic, opt->hin, i_frame + opt->i_seek )) 是从输入文件读数据，被屏蔽
//          break;
            frame = p_get_frame()-1;
//上一行 p_get_frame 函数将采集结束的视频数据缓冲区从
//视频设备驱动程序的缓冲区队列移出，获得已采集结束的视频数据；
//并返回存放视频数据缓冲区首地址的数组下标。
//下面三行代码直接将视频数据缓冲区首地址按 Y（亮度）、U（色度 Cb）、V（Cr）的顺序
//赋值给 X264 程序中原来的视频数据缓冲区地址。从而免除了一次数据拷贝过程，
//提高了程序运行效率。
            pic.img.plane[0] = buffers[frame].start;
            pic.img.plane[1] = pic.img.plane[0] + Ysize;
            pic.img.plane[2] = pic.img.plane[1] + Usize;
//****************** JCM ******************************
            pic.i_pts = (int64_t)i_frame * param->i_fps_den;
            //此处略去若干行未修改的代码
                fflush( stderr ); // needed in windows
            }
//****************** JCM ******************************
            get_next_frame(frame);
//get_next_frame 函数将压缩处理结束的视频缓冲区重新投入视频设备驱动程序的缓冲区队列，
//以备缓冲区的循环使用，继续以后的视频帧数据采集。
//****************** JCM ******************************
    }
    /* Flush delayed B-frames */
    do {
        i_file +=
        i_frame_size = Encode_frame( h, opt->hout, NULL );
    } while( i_frame_size );

    i_end = x264_mdate();
    x264_picture_clean( &pic );
    x264_encoder_close( h );
    fprintf( stderr, "\n" );
//***************************** JCM add ******************************
    stop_capturing();
//停止视频采集与压缩过程，释放内存，关闭视频设备文件等善后处理。
//***************************** JCM add ******************************

    if( b_ctrl_c )
        fprintf( stderr, "aborted at input frame %d\n", opt->i_seek + i_frame );
        //此处略去若干行未修改的代码
    return 0;
```

}

5.3 H.264 编码器的移植和优化

5.3.1 H.264 编码器的移植

移植 x264 编码器是通过使用第三章建立的交叉工具链对 x264 源码进行交叉编译，得到目标平台处理器格式的二进制可执行文件和库文件的过程。H.264 移植主要是两个方面的工作，一是函数库的问题，要移植编译 H.264 所需要的所有函数库。二是修改 Makefile 文件，把里面所有在 X86 上用的编译器都换成交叉编译器进行。

编译器：arm-linux-gcc 4.4.1，重新安装并修改环境变量，将 arm-linux-gcc 编译器指定为 4.4.1：

```
gedit   /root/.bashrc
//在/root/.bashrc 这个文件的最后一行添加上命令：
export PATH =$PATH:/usr/local/arm/4.4.1/bin
```

这样 arm-linux-gcc 4.4.1 安装完毕，就可以使用此交叉编译器了。

对于 x264 源码程序中 CPU 代码需要进行修改，否则会出现关于 cpu_set_t 的错误，关于 cpu_set_t 的以下代码是关于计算 CPU 内核数的代码，可将 np=1(所使用的计算机是单核)直接返回，而没有使用其它的计算。

```
/*#elif defined(SYS_LINUX) unsigned int bit;
int np;
cpu_set_t p_aff;
memset( &p_aff, 0, sizeof(p_aff) );
sched_getaffinity( 0, sizeof(p_aff), &p_aff );
for( np = 0;  bit = 0;   bit < sizeof(p_aff);   bit++ );
np += (((uint8_t *)&p_aff)[bit / 8] >> (bit % 8)) & 1;
return np;*/
```

上面被注释掉的代码可由以下源码代替：

```
#elif defined(SYS_LINUX) int np;
np=1;
return np;
```

首先运行 x264 源码根目录下的 configure 脚本进行配置，如使用“./configure --prefix=/home/x264 –enable-shared –enable-debug”命令进行配置，接着修改 config.mak 文件，在文件中指定交叉编译工具链和编译选项，增加必要的变量定义。修改后的主要选项有：

```
ARCH=ARM
SYS=LINUX
CC=arm-linux-gcc
AS=arm-linux-as
```

.....

这时使用 make 命令编译即可得到 x264 可执行文件以及共享库和静态库文件，然后将该文件移植到开发板上。

5.3.2 H.264 编码器的优化

X264 的源码针对 x86、PowerPC 等平台做了部分算法上和汇编代码级的优化，但并没有针对 ARM 处理器平台进行优化，仅仅通过将 x264 编码技术进行算法优化并移植到 arm 上还远不能满足实际应用的要求，因此必须针对 S3C6410 处理器的特性，结合实际应用的需要，对移植后的程序作进一步的优化。

嵌入式软件的优化涉及到多个方面，往往需要根据嵌入式平台的硬件特性，从开发工具的选择、系统设计、编程开发等角度对程序进行优化。本文针对 S3C6410 处理器与软件优化相关的硬件特性，从编译器的角度和程序设计的角度对 x264 分别进行了编译优化、代码级优化和存储器访问优化。

1) 编译优化

目前很多编译器都实现了较强的代码优化功能，多数编译器都是基于数据流以实现别名分析，常数折叠，冗余代码的消除，循环展开等与体系结构无关的优化，比如本系统所使用的 GNU GCC，GCC 中所采用的-O0、-O1、-O2、等选项选择针对速度/面积等性能进行优化，编译优化属于静态优化，由编译器自动完成，但是对于程序的语义信息，算法流程等信息编译器却很难得到，故需要手动编程优化以提高运行效率。

下面对一些经常用到的主要相关选项[44]进行说明：

(1) -g：全符号调试。生成的代码包含了很多符号信息，程序效率较低，但可调试程序。

(2) -O0、-O1、-O2、-O3、-O4：后面的数字越大，代码性能优化的程序越高。

(3) -O3：除了涉及空间和速度交换的优化选项，执行几乎所有的优化工作，一般在编译时应该使用这个选项。但是在个别情况下，使用这个选项在优化循环，组织流水线的时候可能会出现错误。如果有这种现象出现可以同时使用-g 选项，程序优化就不会出现错误，但是优化效果会下降。

(4) -funrooll-loops：执行循环展开优化，仅对循环次数能够在编译或运行时确定的循环实行。

(5) -fexpensive-optimizations：执行一些相对开销较大的次要优化。

(6) -fschedule-insn：如果对目标机支持这个功能，它试图重新排列指令，以便消除因数据未绪造成的执行停顿。这可以帮助浮点运算或内存访问较慢的机器调取指令，允许其他指令先执行，直到调取指令或浮点运算完成。

2) 代码级优化

H.264 编码器的算法复杂度很高，x264 等软件编码程序设计之初都是以 PC 机作为运行平台，程序代码结构和数据的安排多从实现的难易上考虑。而从嵌入式设备中的应用程序受到处理器和总线频率、内存等硬件资源的限制，为了在嵌入式 PXA255 处理器上实现软件编码器，必须对流程和数据组织从处理器架构、硬件资源和代码运行效率上作详细的考虑。

(1) 去除冗余程序代码

X264 功能强大，支持 H.264 标准中基本档次、主要档次和高级档次的大部分编码工具，视频源数据可以是 yuv 原始视频、avi 或 avs 等格式视频数据，并提供友好的命令行操作接口。但对于本系统的编码方案，只需处理采集的 yuv 原始视频和使用 x264 的小部分编码工具，多余的分支和功能仅会增加代码尺寸，并且降低分支预测的命中率，增加分支判断的开销。因此可以去掉大量不必要的分支跳转或条件判断。为了节省参数分析时间，可以把不必要用户设置的编码选项如运动估计宏块搜索方式、搜索范围、亚像素插值等参数设置为 x264 的缺省设置并保存，而 CABAC、B 条带等不必要的编码工具可以去掉。通过去除 x264 中的冗余程序代码，可简化程序结构，减小代码尺寸，提高程序的运行效率。

(2) 运用内联函数与数据打包技术

Intel 和 GCC 的编译器提供了许多 intrinsics 内联函数[45]，可快速优化 C 代码。intrinsics 是能直接与汇编指令相映射的在线函数，不易用 C 实现其功能的汇编指令都有相对应的 intrinsics 函数。文献[46]中提供了编译器提供的内联函数表，intrinsics 内联函数用下划线“_mm”特别标示，其使用方法与普通的 c 调用函数一样，intrinsics 内联函数为优化程序代码提供了方便快捷的手段，不需要编写专门的汇编源码文件，可以在 C 语言代码中直接嵌入内联函数，达到与汇编指令相类似的效果，如数据打包指令，一条指令即可同时操作多个字节的数据。

在为了使用 intrinsic 内联函数对应的汇编指令中，常要用的多媒体协处理器的 64 位寄存器。在用 intrinsics 内联函数对函数优化时应把_m64 类型变量控制在 16 个内，在写 intrinsics 优化程序时有意识规划好写汇编时 16 个 64 位的寄存器的使用。此外，_m64 类型变量的生命周期内不要调用含有_m64 类型的局部变量的函数，也不要把_m64 类型变量作为函数的参数传递(除 intrinsics 内联函数外)。

3) 存储器访问优化

随着处理器主频的不断提高，处理器与主存之间的速度差异越来越大，存储器访问优化成为性能优化非常关键的一个方面。对于对大量数据处理和计算的应用程序，存储

器访问优化不好，会使 TLB[47]未命中率和 Cache 未命中率很高，Xscale 流水线会经常处于停滞状态等待 Cache 填充，使系统性能很低。

Cache 的优化技术包括以下几个方面：

(1) 借鉴来自高性能计算领域的数据分块、数组合并、循环交换、循环融合、结构重排等程序并行化技术，提高数据的空间局部性和时间局部性，在 X264 的代码中处理的流程是逐宏块进行的，即对于一个宏块的处理流程为运动估计及补偿、变换、量化和熵编码。为了对 Cache 优化，将整个编码流程分为不同的功能模块，每个模块对整帧进行处理，即对整帧图像逐宏块进行运动估计及补偿，再对整帧原始图像和预测图像得到的差值进行变换编码。如在对半像素点插值时，采用整帧插，而 1/4 像素差值算法用相对简单的双线性差值，在 S3C6410 平台上只需一条指令(wavg2rb)就能实现 8 个点的双线性插值，1/4 插值用到整像素平面和半像素平面的不同组合，因此在进行运动估计时，对半像素进行运动搜索完后产生最优半像素点，对周围的 1/4 像素进行实时插值。这样，既提高了代码的时间局部性，又提高了数据的时间与空间的局部性。

(2) 减少处理器对内存的访问，减少数据的存储与读取以及函数的调用。如 H.264/AVC 中采用的是 4×4 整型变换和 ZigZag 扫描，使用 S3C6410 中多媒体协处理器单元提供的 64 位寄存器，对 4×4 整型变换采用蝶形变换进行并行的处理，对于 4×4 整型变换结果存储在 4 个 64 位的寄存器中再对这些保存在 64 位寄存器里的数据直接进行打包等操作完成 ZigZag 扫描。

(3) 选择合适的数据类型，增加 WMMX 指令的并行度，从而减少编码器的核心运算量。在视频编码系统中的变换编码部分，H.264/AVC 采用的是 4×4 整型变换。将数组 x[]定义为 short 型，代码的执行速度能提高 2 倍多，相应的核心运算所用的周期数、Cache miss 数减半。

(4) 数据储存的地址应该按照 8 字节对齐进行存储，因为 Xscale 提供的多媒体加载指令一次可以加载 8 个字节，但要求数据是 8 字节对齐的。增加数据的存储速度有利于提高整个编码器的速度。

(5) 针对在 S3C6410 处理器指令集中，提供了数据预装指令 PLD，PLD 指令能预装一个 Cache line 的数据到 Cache 中，以便给下面的执行程序使用，有效的降低了 Cache miss 的发生，如果预载的数据已在 Cache 中，PLD 指令将不执行任何操作。PLD 预载的数据要等待若干个周期(一般为 8 个周期以上为佳)才能到 Cache 中，这是因为存储器和 CPU 的速度的不同，从内存读取数据到 Cache 的周期数也不相同，所以，应在程序中提前 8 个周期以上或当前循环为下一循环用 PLD 指令预载数据。在 PLD 预装数据到 Cache 中时，CPU 可以照常运行，实际上如上所说类似于 DMA 机制，很好的运用 PLD

指令，能够有效的降低 Cache miss 的发生。

5.4 本章小结

本章介绍了嵌入式 Linux 视频采集相关的 USB 子系统、驱动程序的结构，在详细阐述了基于 V4L2 的 ioctl 系统调用与编程模型的基础上，实现了一个基于 V4L2 的采用内存映射机制的视频采集程序，并提出了基于 x264 和 Linux 中 V4L2 的视频采集与压缩程序设计思路。最后将 X264 项目源码与视频采集程序合并，针对 S3C6410 平台进行编译器的优化，设计实现了视频采集压缩一体化的视频采集编码模块。

第六章 视频传输模块设计

由于数字视频传输的信息量巨大和传输带宽有限，无线视频传输技术需要解决视频编码数据包的可靠性传输，低码流传输，实时解码等问题。网络传输协议的选择、传输控制策略的设计等成为视频在网络传输中的关键技术。

6.1 网络传输控制协议栈

数据在网络上传输需要遵守一定的网络协议，数据的类型决定了网络协议的选择。针对网络上传输音视频等多媒体数据的需要，电信行业和计算机业很多国际组织进行了许多标准化的工作，目前较为关键的协议有 TCP/IP、UDP、RTP/RTCP、RTSP 等。本系统传输部分主要采用 RTP/RTCP/UDP 协议。

在 TCP/IP 参考模型中，传输层通信协议 TCP 和 UDP 均不能满足视频传输的 QOS 要求。TCP 协议采用滑动窗口控制协议机制，数据传输随着流控窗口动态的启动和关闭，另外 TCP 重传会造成时延，使其难以满足多媒体实时的传送要求。UDP 协议提供不可靠交付语义，没有重传设施，如果发生故障，不会通知发送方，且 UDP 不提供流控。

为了支持网络实时传输服务，提供网络的实时传输标准，IETF 的视频/音频传输工作小组制订了 RTP 和 RTCP 协议。实时传输协议 RTP(Real-time Transport Protocol)是用于 Internet 上针对多媒体数据流的一种传输协议。实时传输控制协议 RTCP(Real-time Transport Control Protocol)是为 RTP 协议的所有参与者提供 QoS 反馈而设计的伴随协议。RTP 主要实现的是一种端到端的多媒体同步机制，既不需要事先建立连接，也不需要中间节点的参与而为其保留资源。RTP 报文用于传送媒体数据(如音频和视频)，它由报头和数据两部分组成，RTP 的数据部分称为有效载荷。RTP 报文包括负荷类型标识、序列编号、时戳、同步信源标识和特约信源标识等。RTCP 报文用于传送控制信息。RTCP 报文包括丢失率、接收到的最高序列号和同步信源标识等，这些为媒体同步、丢包统计、传输检测和传输复用等手段提供了可能性。

RTP 协议本身独立于下层的传输层和网络层，与下层协议无关，它一般可以和其他底层网络或者传输层协议共同工作。RTP 数据分组没有包含长度域或其他边界，其分组的最大长度仅仅被下层网络限制。RTP 数据通常采用 UDP/IP 封装，它利用 UDP 的多路复用及校验和服务，共同完成数据的传输功能。

实时流协议 RTSP：实时流协议 RTSP(Real-time Streaming Protocol)是应用级协议，用于控制具有实时特性的数据传输，它提供一个可扩展的框架，以使诸如音频和视频之类的实时媒体的点播传输成为可能[48]。RTSP 在体系结构上位于 RTP 和 RTCP 之上，它

使用 TCP 或 RTP 完成数据传输。

监测系统是在传输层协议 (TCP，UDP)上解释 RTP， RTCP， RTSP 协议的，所有的客户连接请求都是以 TCP 的端口获得的，媒体数据也都是打成 RTP 包[49]，通过 UDP 端口发出去的。当监测系统面对一个单一的客户，完成的过程如下：

1) 在客户端发出 RTSP 连接请求后，服务器通过对 TCP 端口的监听，读入请求；

2) 解析请求内容，调入相应的流媒体文件；

3) 形成 RTP 包，分发数据流包，获得 RTCP 包；

4) 数据包发送完毕，关闭连接。

6.2 RTP/RTCP 协议分析

6.2.1 RTP 协议分析

一个 RTP 数据包由 12bytes 的头部信息和有效负载域组成。有效负载域可以携带音频采样序列或视频帧。头部信息域包含信息如图 6.1 所示[48]：

V	P	X	CC	M	PT	Sequence number
Timestamp						
Synchronization source（SSRC） identifier						
Contributing source（CSRC）identifiers						

图6.1 RTP包格式

版本域(V)：2 比特，这个域定义了 RTP 的版本。RFC3550 定义的版本号是 2。

填充位(P)：1 比特，若该比特被设置，此包里包含有一到多个附加在末端的填充比特，而不是负载的一部分。

扩展位(X)：1 比特，若设置扩展比特，固定报头后面跟随一个头扩展。

CSRC 计数(CC)：4 比特，CSRC 计数包含了跟在固定报头后面 CSRC 标识符的数目。

标志位(M)：1 比特，标志的解释由具体协议规定。它用来允许在比特流中标记重要的事件，如帧边界。

负载类型(PT)：7 比特，这个域定义了负载的格式，协议可以规定负载类型码和负载格式之间的一个默认匹配[50]。其他的负载类型码可以通过非 RTP 方法动态定义。

序列号：16 比特，每发送一个 RTP 数据包，序列号就增加 1。接收方可以依次检测数据包的丢失并恢复数据包序列。

时间戳：32 比特，时间戳是在 RTP 数据包的第一个字节采样时刻[51]。采样时刻必须随时间单调线形增加，以允许同步和抖动计算。时钟分辨率必须满足要求的同步精确度，以及对包到达的抖动的测量，并且可以分解成端到端延迟的分辨率。时钟频率与作

为载荷携带的数据格式无关。对于周期性发送 RTP 包而言，采样时刻是由采样时钟确定的。如果一个应用每读入一个数据块要占据 n 个采样周期，则无论数据块被发送或丢失，每个时间戳都会增加 n。

同步源 SSRC：32 比特，SSRC 标志同步信源，数值是随机选取的，且在同一个 RTP 会话中没有两个发送者的 SSRC 标志相同。

6.2.2 RTCP 协议分析

与 RTP 协议相配的控制协议是 RTCP。RTCP 包把关于会话质量的端到端信息传送给每一个参与者，像包延迟、抖动、收到和包丢失等数值对网络而言非常有价值，可以用来实时估计网络自身的状态。有 5 种类型的 RTCP 包：

SR：Sender Report，	发送者报告
RR：Receiver Report，	接受者报告
SDES：Source DEScription，	信源说明
BYE：Hang up from a session，	挂断会话
APP：Application-Specific acket，	特定应用的包

系统中并不是 5 种 RTCP 报文都必须实现，其中 SDES 报文主要功能是作为媒体会话成员标识信息的载体，此报文的功能在本视频传输方案中是多余的，另外，APP 报文目的是测试新开发的应用和特征，这里也不需要。RTCP 报文由公共包头部分和结构化的内容组成，报文内容根据报文类型的不同而具有不同的长度，一般以 32 位为边界。

V	PT	报文长度
SSRC 标识符		
NTP 时戳（高字节部分）		
NTP 时戳（低字节部分）		
RTP 时戳		
发送者的报文计数		
发送者有效载荷字节计数		

图6.2 SR报文格式

6.3 基于 RTP 的 H.264 视频流的传输控制方法

RTP 协议仅仅实现了网络传输层的功能，要实现媒体数据的网络传输，网络层和会话层协议也必不可少，在传输层，为实现真正的端对端传输，RTP 必须以 UDP 或 TCP 为底层协议；用 RTSP(Real-Time streaming Protocol)协议完成会话控制；在网络层，IP 协议完成网络寻址等最基本的网络层功能。图 6.3 为整个监测系统的传输控制协议栈。

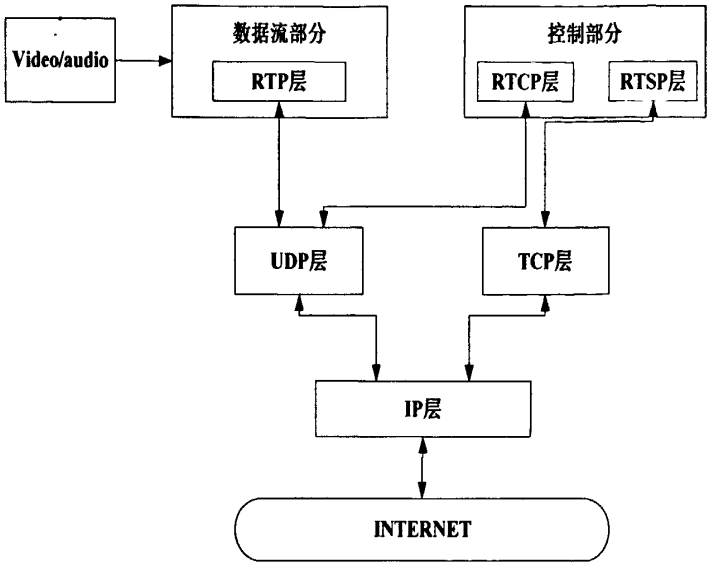


图6.3 无线监测系统传输控制协议栈

在数据平面，服务器端将压缩打包后的音视频数据按照 RTP 数据传输协议的报文格式装入 RTP 报文的数据负载段，同时配置 RTP 报文头部的时间戳、同步信息、序列号等重要参数，此时的数据报文已经具有典型的时间特征，即被“流化”了。在 UDP/TCP 层，RTP 报文作为负载数据装入 UDP 报文中后，由 IP 层负责最后的报文头部配置，实现报文的路由选择与网络传输。在控制部分，RTCP 和 RTSP 报文通过 UDP/TCP 层后，同样由 IP 层负责转发。RTSP 的主要功能是实现停滞、暂停、快进等 VCR 控制操作，RTCP 仅负责控制 RTP 报文的传输。RTP 数据传输协议负责音视频数据的流化和负载，RTCP 负责 RTP 数据报文的传输控制。并通过 RTP 协议与 RTSP、UDP/TCP 等协议组成一个完整的网络体系，最后由 socket UDP 完成视频流的传输。

基于以上分析，本论文使用 RTP 来打包传送 H.264 视频流，使用 RTCP 来实现拥塞控制和流量控制服务。其中 RTP/RTCP 协议是基于 UDP 协议进行数据传输，应用层由 socket 套接字网络程序完成。基于 RTP 协议的 H.264 视频传输控制框架[50-51]如图 6.4 所示：

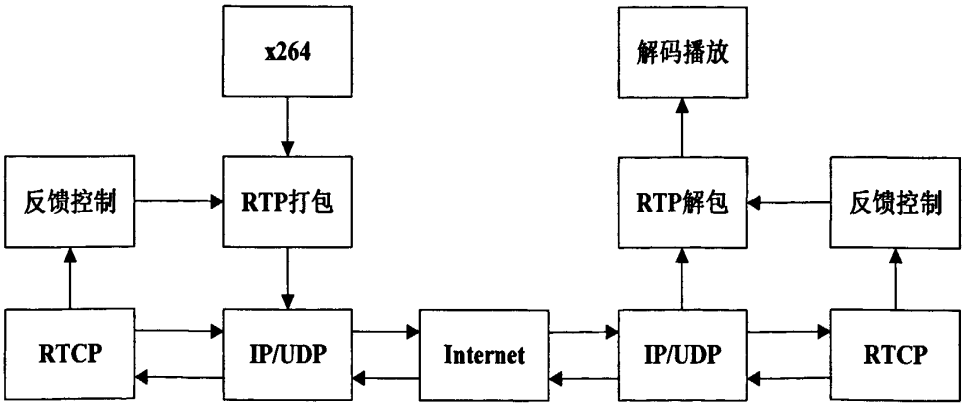


图6.4 基于RTP协议的H.264视频传输控制框图

6.4 H.264 视频流的 RTP 封包策略

采用 RTP 传输 H.264 视频流，需要对其进行封包，加入时间戳、序列号等信息。将 H.264 封包一般有三种封装模式：

单一模式(Single NAL unit mode)，即把一个 NALU 封装成一个 RTP 包，该方法主要适用于对话应用系统；

非交错模式(Non-interleaved mode)，按照编码出来的视频流的顺序进行封包，此方法主要适用于低延迟的实时系统；

交错模式(Interleaved mode)，将打乱编码出来的视频流顺序进行封包，主要适用于对延迟要求较低的系统。

一般 H.264 采用简单封包的方案，即一个 RTP 分组里放入一个 NALU(NAL 单元)，编码器接口产生的 H.264 视频流 NALU[52]分别被封装上 RTP 包头、UDP 包头和 IP 包头，然后 IP 数据包通过 3G 无线传输网络传输给接收端。经封装后的包格式如图 6.5 所示：

IP	UDP	RTP	NAL unit

图6.5 NAL单元的封装格式

接收端收到 IP 包后按相反的顺序将 RTP 包头和视频数据提取出来，根据 RTP 包头的序列号进行排序后将视频数据通过解码器接口传入解码器，解码器对数据解码后进行视频播放。

6.5 视频传输控制模块的实现

RTP 协议是目前解决流媒体传输问题的非常好的方法，如果需要在 Linux 平台上进行流媒体编程，可以考虑使用一些开放源代码的 RTP 库。目前有很多开放的源码库提供网络协议 RTP/RTCP 功能的实现，比较流行的有 LIBRTP、JRTPLIB 等。JRTPLIB 是一个面向对象的 RTP 库，它完全遵循 RFC1889 设计，使用 socket 机制实现网络通讯，因此可以运行在 Windows、Linux、FreeBSD、Solaris、Unix 和 Vxworks 等多种操作系统上。由于视频传输部分较为复杂，本文仅对传输核心部分进行详细阐述，下面是 JRTPLIB 在嵌入式 Linux 平台上运用 RTP 协议进行设计编程的具体实现过程。

6.5.1 环境的搭建

为 Linux 系统安装 JRTPLIB，从 JRTPLIB 网站下载最新的源码包，此处使用的是 jrtplib-3.4.tar.gz，下载后的源码包保存在/tmp 目录下，执行下面得命令可以对其进行解压缩：

```
#cd /tmp
#tar -zxvf jrtplib-3.4.tar.gz
```

对 jrtplib 进行配置和编译：

#cd jrtplib-3.4

#./configure CC=arm-linux-g++ cross-compile=yes

修改 Makefile 文件，将连接命令 ld 和 ar 改为 arm-linux-ld 和 arm-linux-ar

#make

#make install

6.5.2 基于 RTP 的 JRTPLIB 编程

将 JRTPLIB 动态库交叉编译成可运行在 ARM+Linux 操作系统中的版本后，通过用户程序调用 JRTPLIB 提供的接口函数实现媒体数据的发送传输、接收程序，利用 JRTPLIB 函数设计服务器端 RTP，RTCP 发送，RTCP 接收线程。

1) 初始化 RTP 会话

在使用 JRTPLIB 进行数据传输之前，应首先生成 RTPSession 类的一个实例来表示此次 RTP 会话。TCP/IP 的通信是通过套接口(sockct)来实现，而 RTP/RTCP 是通过一个会话 (session)来实现的。通过调用 Create()方法可以对其进行初始化操作，RTPsession 类的 Create()方法有一个参数，用来指明对话的端口号。

例如：RTPSession sess;

sess.Create(5000);

如果 RTP 会话创建失败，create()方法将会返回一个负数，因此虽然可以较容易判断函数调用究竟是成功还是失败，但却很难明白具体的出错的原因。JRTPLIB 采用了一种统一的错误处理机制，它提供的所有函数如果返回负数就表明出现了某种形式的错误，而具体的出错信息则可以通过调用 RTPGetErrorString()函数得到。RTPGetErmrString()函数将错误代码作为参数传入，然后返回该错误代码所对应的错误信息。下面给出了完整的初始化框架，它可以对 RTP 会话初始化过程中所产生的错误进行很好的处理：

```
#include<stdio.h>
#include “rtpsession.h”
int main(void)
{
RTPSession sess;
int status;
char *msg;
sess.Create(6000);
msg=RTPGetErrorString(status);
printf(“Error String:%s\\n”, msg);
return 0;
}
```

设置恰当的时间戳单元，是 RTP 会话初始化过程所需要进行的一项重要工作，它是

通过调用 RTPsession 类的 SetTimestampUnitO 方法来实现的，该方法同样也只有一个参数，表示的是以秒为单元的时间戳单元。例如，当使用 RTP 会话传输 8000Hz 采样的音频数据时，由于时间戳每秒钟将递增 8000，所以时间戳单元相应地应该被设置成 1/8000：sess.SetTimestampUnit(1.0/8000.0);

2) 发送 RTP 数据流

RTP 会话成功的建立起来后，接下来就可以开始进行媒体数据的传输了。首先设置好数据发送的目标地址，RTP 协议允许同一个会话存在多个目标地址，这可以通过调用 RTPsession 类的 AddDestination()、DeleteDestination()和 ClearDestinations()等函数方法来完成。下面的语句表示的是让 RTP 会话将数据发送到本地主机的 6000 端口：

```
Unsigned long addr=ntohl(inet_addr("127.0.0.1"));
sess.AddDestination(addr, 6000);
```

目标地址全部指定之后，接着就可以调用 RTPSession 类的 SendPacket() 方法，向所有的目标地址发送数据。SendPacket() 是 RTPSession 类提供的一个重载函数，对于同一个 RTP 会话来讲，负载类型、标识和时戳增量通常是相同的，JRTPLIB 允许将它们设置为会话的默认参数，这是通过调用 RTPSession 类的 SetDefaultPayloadType()、SetDefaultMark()和 SetDefaultTimeStampIncrement() 方法来完成的。为 RTP 会话设置这些默认参数的好处是可以简化数据的发送，例如，如果为 RTP 会话设置了默认参数：

```
sess.SetDefaultPayloadType(0);
sess.SetDefaultMark(false);
sess.SetDefaultTimeStampIncrement(10);
```

之后在进行数据发送时只需指明要发送的数据及其长度就可以了：

```
sess.SendPacket(buffer, 5);
```

3) RTP 数据流的接收

对于数据的接收端，首先要调用 RTPsession 类的 PollData()方法来接收发送过来 RTP 或者 RTCP 数据报。因为同一个 RTP 会话允许有多个参与者，既可以通过调用 RTPsession 类的 GotoFirstSource()和 GotoNextSource()方法来遍历所有的源，也可以通过调用 RTPSession 类的 GotoFirstSourceWithData()和 GotoNextSourceWithDataO 方法来遍历那些携带有数据的源。在从 RTP 会话中检测到有效的数据源之后，就可以调用 RTPSession 类的 GetNextPacket()方法从中提取出 RTP 数据报，当接收到的 RTP 数据报处理完之后，要及时进行内存空间的释放，以免发生内存泄漏。下面的语句为对接收到的 RTP 数据报进行处理:

```
if(Sess.GotoFirstSourceWlthData())
{
```

```
do{
RTPPacket*Pack;
pack=sess.GetNextPacket(); //处理接收到的数据
delete pack;
}while(sess.GotoNextSourceWithData());
}
```

JRTPLIB 为 RTP 数据报定义了三种接收模式，其中每种接收模式都具体规定了哪些到达的 RTP 数据报将会被接受，而哪些到达的 RTP 数据报将会被拒绝。通过调用 RTPsession 类的 SetReceiveMode()方法可以设置下列这些接收模式：

RECEIVEMODE_ALL：缺省接收模式，所有到达的 RTP 数据报都将被接受；

RECEIVEMODE_IGNORESOME：除了某些特定的发送者之外，所有到达的 RTP 数据报都将被接受，而被拒绝的发送者列表可以通过调用 AddToIgnoreList()、DeleteFromAcceptList()和 ClearAcceptListO 方法来进行设置；

RECEIVEMODE_ACCEPTSOME：除了某些特定的发送者之外，所有到达的 RTP 数据报都将被拒绝，而被接受的发送者列表可以通过调用 AddToAcceptList()、DeleteFromAcceptList 和 ClearAcceptList()方法来进行设置。

4) 控制信息

JRTPLIB 是一个被高度封装的 RTP 库，在使用它时并不需要关心 RTCP 数据报是如何被发送以及接收的，因为这些都通过 JRTPLIB 库自己来完成。当 PollData()或 SendPacket()方法被成功调用时，JRTPLIB 就能够自动对到达的 RTCP 数据报进行处理，并且在适当的时候反馈 RTCP 数据报给对方，以确保整个 RTP 会话过程的正确性。

通过调用 RTPSession 类提供的 SetLocalName()、SetLocalEmail()、SetLocalLocation()、SetLocalPhone()、SetLocalTool()和 SetLocalNote()等方法，可以对 RTP 会话的控制信息进行设置。所有这些方法在调用时都带有两个参数，第一个参数是一个 char 型的指针，指向将要进行设置的数据，第二个参数是一个 int 型的数值，用于表明该数据的前面多少个字符将会被使用。在 RTP 会话过程中，不是所有的控制信息都需要被发送，通过调用 RTPSession 类提供的 EnableSendName()、EnableSendLocation()、EnableSendPhone()、EnableSendTool()和 EnableSendNote()方法，可以为当前 RTP 会话选择将被发送的控制信息。

关于上层 socket 网络程序属于纯软件开发部分，本文在此不再阐述。

6.6 本章小结

本章通过研究分析 RTP/RTCP 协议，提出了基于 RTP 的 h.264 视频传输控制框架设计，通过分析对 H.264 视频流进行 RTP 封包，在嵌入式 linux 下采用开放的面向对象的 RTP 源码库 JRTPLIB 完成对传输控制模块的实现。

第七章 测试与分析

系统的调试与测试工作是工程开发与研究至关重要的一步，本章对监测终端进行测试分析。

7.1 3G 无线网络的接入

通过 3G 模块链接 3G 网络的过程是通过 PPP 协议完成的。而 Linux 下的 PPP 拨号上网可以调用 PPPd 和 chat 这两个应用程序，经过 AT 指令来实现。

PPPd 的全称是 Point to Point Protocol daemon，即点对点协议后台程序。PPPd 提供了基本的 LCP、认证、NCP 的支持，建立和维持与服务器的 PPP 链接，并传输数据。Chat 程序的用途是拨号，并等待提示，根据提示输入用户名、密码等登录信息。

在嵌入式 Linux 平台上通过 3G 模块拨号上网需要经过以下几个步骤：

1) 配置 Linux 内核支持 PPP

执行 make menuconfig 配置内核时，选择 network device support，配置 kernel 使它支持如下的 PPP 选项：

PPP (point-to-point) support
PPP multilink support (EXPERIMENTAL)(NEW)
PPP support for async serial ports(NEW)
PPP support for sync tty ports(NEW)
PPP deflate compression(NEW)
PPP BSD-Compress compression(NEW)
PPP over Ethernet(EXPERIMENTAL)(new)

以上选项主要是 PPP 协议、PPP 同步/异步串口通信、PPP 压缩等功能。重新编译内核后，此时生成的 Linux Kernel 便支持 PPP 了。

2) 安装和配置 PPP 拨号工具。将 pppd 源码 ppp-2.4.5.tar.gz 解压到某个目录下，执行：

```
export PATH=/usr/local/arm/4.4.1/bin/:$PATH
export CC=arm-linux-gcc
cd /root/ppp-2.4.5
./configure
make
```

3) 编写拨号脚本

Linux 系统中，3G 拨号采用 shell 脚本实现，链接中国移动的 3G 网络，在/etc/ppp/peers/目录下创建一个 ChinaMobile 文件，内容为：

```
ttySAC1                    //3G 连接到/dev/ttySAC1 串口上
115200                     //串口波特率
Crtscts                    //rts 和 cts 信号线用于流控
debug                      //PPPd 工作在调试模式
nodetach                   //不转为后台进程
ipcp-accept-local          //接受分配的本机 IP 地址
ipcp-accept-remote         //接受指定的服务器 IP 地址
defaultroute               //将服务器 IP 地址作为默认的路由
user card                  //认证时的用户名为 card
```

以上是对/etc/ppp/peers/ChinaMobile 的说明，另外还需要的一个文件是/etc/ppp/cdma-chat，他的作用是告诉 chat 程序如何完成建立连接的过程，该文件的主要内容是：

```
ABORT"NO CARRIER"
ABORT"NO DIALTONE"
ABORT"ERROR"
ABORT"NO ANSWER"
ABORT"BUSY"
TIMEOUT 120
""at
OK atdt#777
CONNECT
```

几个 ABORT 行的意思是：如果 modem 返回诸如 BUSY 之类的信息，则取消 chat 过程，连接失败；TIMEOUT 120 表示连接的超时值为 120 秒。首先发出一个 at 命令，期待 modem 返回一个 OK，然后拨号#777，期待 modem 返回 CONNECT，如果 modem 返回了 CONNECT，则表明连接建立成功。

7.2 测试项目和分析

就目前的 3G 网络状况来讲，实现实时数据业务还存在掉线率高、网络延时大、容易发生网络拥塞等不足，而且在应用中发现数据传输数率很难达到理论上的最大数率。理论上 3G 的最大速率可达 2Mbps，但实际的传输速率是 400Kbps 左右。

系统调试连接装置如下：

网络视频采集设备硬件 S3C6410 嵌入式开发平台，256M 的 NANDFLASH，操作系统为移植的 ARM-Linux，配接 USB 接口的摄像头进行工作。进行测试的设备，一个是开发平台本身，在开发平台上使用 3G 模块接入 3G 网络，还有就是本地宿主 PC 机和远

程监控 PC，配置：Pentium4 3.0G 内存 1G，操作系统是 RedHat Linux9.0。

1) 测试项目

(1) 编码器性能测试

H.264 编码器经过算法级优化及针对 S3C6410 平台优化后性能得到了一定的改善，优化后，编码帧率可以达到 23.76，编码速度最快的 akiyo 编码帧率可达到 29.98，达到了实时的效果。论文采用标准视频序列进行测试，下表显示了优化后编码器的性能测试结果：

表7.1 H.264编码器性能测试结果

视频序列	格式	速率（fps）	PSNR(db)	码率(kbit/s)
Akiyo	QCIF	29.98	39.857	32.26
monitor	QCIF	24.13	37.446	59.65
Carphone	QCIF	23.76	37.832	118.82

图 7.1 为 akiyo 视频序列原始图像，图 7.2 为 akiyo 视频序列编码后图像。通过图示可以看到，编码后的的图像效果与原始图像效果比较接近。

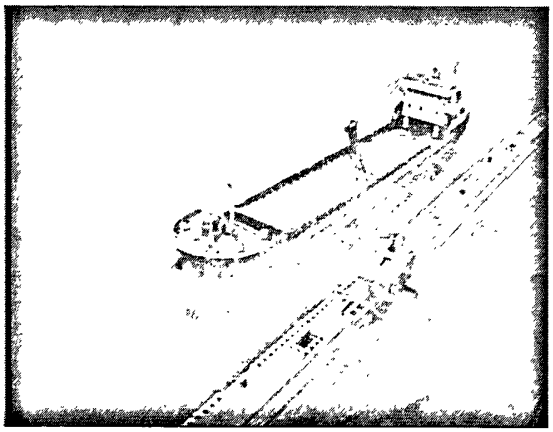


图7.1 akiyo视频序列原始图像

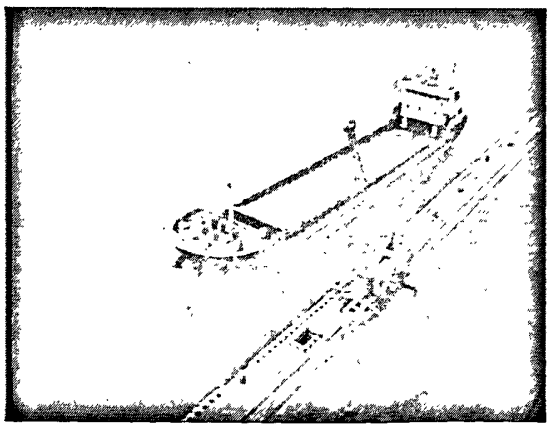


图 7.2 akiyo视频序列编码后图像

(2) 服务器端通过网络接收 H.264 的 RTP 包后，经 MPlayer 播放器播放效果质量如

下图 7.3(a)、图 7.3(b)所示：

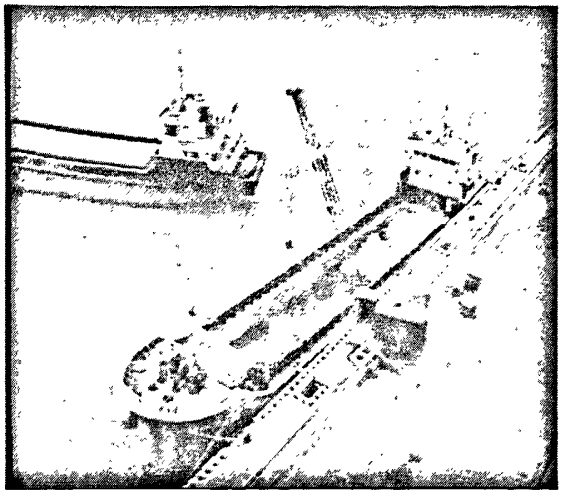


图7.3(a) 接收端MPlayer播放效果图

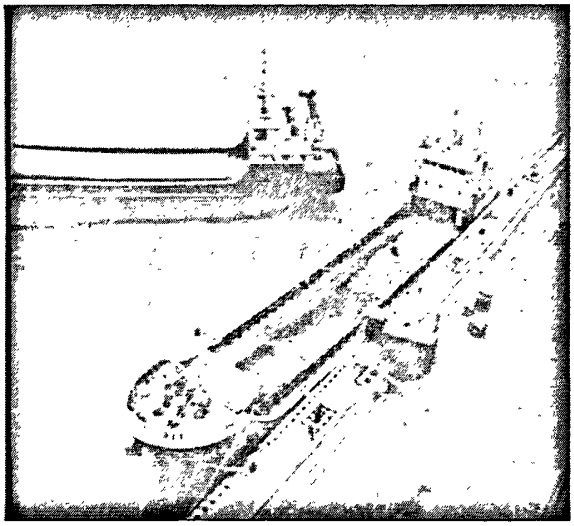


图7.3(b) 接收端MPlayer播放效果图

2) 经过测试分析，可得出以下结论：

(1) 经过优化过后的编码器性能得到了很大改善，基本满足嵌入式应用的准实时要求。监测终端采集的视频格式为 YUV420 格式，实现了将 YUV420 格式的 QCIF 原始图像编码为适合无线网络传输的视频流。

(2) 监测终端采集视频信息到服务器接收端获得视频图像的时间间隔较长。这一方面由于 Linux 本身并不支持实时性；这一点，可通过使用 RTLinux 的 patch，从而增强 Linux 的实时性；另一方面是由于编码延迟和网络延迟，这一部分可以通过研究对编码进一步优化同时采用最新的网络技术，如 3G 等。

(3) 本系统设计基本达到实时监测系统的要求，由于编码延迟和网络延迟，本监测

终端适合对延迟要求不高的应用场合。

7.3 本章小结

本章主要完成了嵌入式开发平台上的3G连接测试，首先给出3G网络接入的原理，然后完成x264编码器优化过后的性能测试及监控效果测试；结合软硬件平台本身及3G无线网络的带宽，基本达到了实时的效果，正在不断完善之中，但是本文探索性的工作已经完成。

第八章 总结与展望

8.1 总结

近年来视频监测得到的迅速发展，应用范围越来越广。随着视频信息处理技术的改进和提高，视频监测从传统有线视频监测进入到了无线视频监测时代，嵌入式视频监测系统具有体积小、成本低、可靠性高、使用方便等优点，有着广泛的应用前景。

本文的研究课题是海事视频监测系统的研究和设计，在经历了一年多时间的毕业设计后，我深刻意识到无论是从工程应用还是科学研究的角度来讲，无线海事视频监测的研究设计都是一个非常复杂的大课题，它和当今世界信息技术的发展紧密相连，融合了传统的视频监测和现代信息技术为一体。每一项技术都凝住了全世界诸多科研工作者的心血，仅靠个人能力是不可能完成全部的理论和技术研究。

下面对课题的主要工作进行一个总结：

1) 论文以海事视频监控作为应用对象，围绕如何实现无线视频监测系统，设计出一种基于 3G 网络以嵌入式 Linux 和 S3c6410 微处理器等平台为基础的无线视频监测系统方案。

2) 详细阐述了嵌入式Linux系统的构建过程，包括交叉编译环境的建立、BootLoader的移植、Linux 内核编译与移植、根文件系统的设计等。

3) 对 H.264 编码中耗时较多的运动估计算法做了深入的研究，提出了一种快速运动估计算法，针对帧内预测模式选择算法中 4×4 块的最优预测进行优化处理，降低编码复杂度，提高编码效率。

4) 选取 ZC0301 芯片的摄像头作为本论文的视频采集设备，根据 Linux 内核提供的 Video4Linux2 的 API 功能函数，实现了视频采集的功能。同时对采集和编码程序进行了融合，并针对 S3C6410 进行优化，完成视频采集编码模块。经过优化后编码器的编码速度达到了 23.76fps，达到了了嵌入式应用的实时性。

5) 提出基于 RTP 的 H.264 视频传输控制方法。对 H.264 视频流的 RTP 封包策略进行了研究，并利用 JRTPLIB 库，实现了一个基本的 RTP/RTCP 构架，最后通过 socket UDP 网络通信完成 H.264 视频流的传输。

8.2 展望

本文系统在以下几个方面仍有待于继续研究和完善：

1) 在 H.264 的优化部分要加强。H.264 的优化是无止境的，进一步的优化能够使编

码效果得到提升，另外针对 S3C6410 平台的优化仍需要进一步改进。

2) 在海事视频监测系统研究设计中，本次工作没有涉及模式识别和数据挖掘等技术，为了更好的起到预警作用，在后续的研究设计中应该将这些技术用到海事视频监控中。

3) 扩展系统的功能，增加对音频信号的监控。因为系统采用的 RTP/RTCP 传输协议支持音视频的传输，所以这一方案无论是从硬件还是软件角度来考虑都是可行的。

4) 本系统最终监测基本达到实时监测的效果，可通过使用 DSP 编码芯片让该系统的性能得到进一步的改善。

参考文献

[1] Joint Video Team (JVT) of ISO/ IEC MPEG and ITU-T VCEG, Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H. 264/ISO/IEC 14496-10 AVC), JVT-G050, 2003.

[2] Yang J F, Chang S C, Chen C Y. Computation reduction for motion search in low rate video coders [J]. IEEE Transactions on Circuits and Systems for Video Technology, 2002.

[3] 洪晶 海事场景的视频监控系统 浙江大学 2007.6

[4] H.264/MPEG-4 Part 10 White Paper.2007.10.2. fttp://www.vcodex.com

[5] Geer.D.Survey:Embedded Linux Ahead of the Pack[J].Distributed Systems Online,IEEE,2004,5(10):3-3.

[6] Araujo.Guldo，Centodueatte.Paulo，Azevedo.Rodolfo.ExPression-tree-based algorithms for eode compression on embedded RISC archltectures[J].IEEE Transactions on Very Large Scale Integration Systems，2000，8(5):530-533.

[7] T-Y Kuo, C H Chan, Fast variable block size motion estimation for H.264 using likelihood and correlation of motion field, IEEE Transactions on Circuits and Systems for Video Technology, 2006.

[8] http://iphome.hhi.de/suehring/tml/.

[9] http://www.videolan.org/,http://forum.videolan.org/.

[10] http://nchc.dl.sourceforge.net/sourceforge/t264/.

[11] 李世平,H.264 三大开源编码器之测评报告.

[12] (美)MattWelsh…等.LINUX 权威指南(第三版).北京:中国电力出版社，2000

[13] http://www-128.ibm.com/developrworks/cn

[14] 毛德操，胡希明.LINux 内核源代码情景分析.杭州:浙江大学出版社，2001

[15] http://www.busybox.net

[16] H.264/MPEG-4 Part 10 White Paper.2007/10/2 .fttp://www.vcodex.com

[17] 余兆明，查日勇，黄磊等.图像编码标准 H.264 技术[M].北京：人民邮电出版社，2006：6

[18] 沈兰荪，卓力. 小波编码与网络视频传输[M]. 北京：科学出版社，2005：233～236

[19] ThomasWiegand，Gary J.Sullivan,Gisle Bjontegaard，and Ajay Luthra.Overview of the H.264/AVC Video Coding Standard[J].IEEE Transactions on Circuits and Systems for VideoTechnology. 13(7).2003：570～576

[20] Iain E.G.Richardson.H.264 and MPEG-4 Video Compression [M].England，John Wiley&Sons Ltd，2003：187~191

[21] PRATT WK，KANEJ.and ANDREWS，HC Hadamard transform image coding[J]，IEEE Proc.57 l(Jan.1969):58-68.

[22] 余磊; 张根栋; 张尧; 赵永政; H.264 中 CABAC 算法与 CAVLC 算法比较和改进 新通信, 2009 年 11 期

[23] 李东江; 唐义平; H.264 中 CAVLC 的分析与实现 网络安全技术与应用 2008 年 01 期

[24] A.Joch，F.Kossentini and P.Nasiopoulos. A performance analysis of the ITU-T Draft H.26L video coding standard[EB/OL]: http://research.microsoft.com/workshops/pv2002/papers /35-nteueubjoc.pdf. 2002-4-25

[25] 黄晃,沈燕飞,李锦涛等,降低视频编解码复杂度的算法研究 中国数字音视频编解码技术标准工作组(AVS)第四次会议文档 AVS_M1056:武汉:2003.3.

[26] Looijenga L Biemond J， Boekee D E et al. A pelrecursive wiener-baed displacement estimation

algorithm[C]. 1987，Sign Proc, 13：399~412

[27] Nandhakumar N Aggarawal J K. On the computation of motion from sequences of images[J]. Proc IEEE，1988.76(8)：917~935

[28] Heitz F. Bouthemy P.Motion estimation and segmentation using a global Bayesian approach[A]. IEEE Pro Int Conf ASSP Albuquerque NM[C]. 1990，1：2305~2308

[29] Bierling M. Displacement estimation by hierarchical blockmatching[J]. SPIE Visual Communications and Image Processing，1988，1001：942~951

[30] 刘丹.计算机图像处理的数学和算法基础[M].国防工业出版社，2005，7：105~108

[31] TekalP A M.Digital Video Processing.Prentice Hall，Inc.1995

[32] 赵波.视频编码中运动估计技术的研究与实现[D].西安：西安电子科技大学硕士学位论文,2005

[33] CeZhu，XiaoLin，and LaP-Pui Chau.Hexagon-Based Search Patten for Fast Block Motion Estimation，IEEE Trans on CSVT，PP.349 一 355，Vol.12 No.5，May，2002

[34] 罗智勇.针对视频会议应用的 H264 视频编码器研究与实现[D].广州：暨南大学.2005

[35] 朱林，冯燕.基于单指令多数据技术的 H.264 编码优化[J].计算机应用. 2005, 25(12)：2798~2802

[36] Lifeng Song，GangWei. JVT-D0l6.doc，Joint VideoTeam(JVT)of IS0/IEC MPEG&ITU-T VCEG (ISO/IEC JTCI/SC29/WGI1 and ITU-T SG16Q.6)4 Meeting:Klagenfurt，Austria，22-26July，2002

[37] H-Y.C.TouraPis，A.M.TouraPis，P.ToPiwala. Fast Motion Estimation within the JVT codec，JVT 一 E023，5th Meeting:Geneva，Switzerland 09-17October，2002

[38] 祝世平; 申晓东; 十字交叉六边形块运动的估计搜索 光学精密工程 2009 年 12 期

[39] 于新波，视频压缩中的预处理及运动估计算法的研究.硕士学位论文.山东大学.2008

[40] Tai, Shen-Chuan; Chen, Ying-Ru; Chen, Yu-Hung; Small-diamond-based search algorithm for fast block motion estimation Source: Signal Processing: Image Communication,v22,n10,p877-890,November 2007

[41] JONATHAN CORBET ALESSSANDRO RUBINI&JONATHAN CORBET. LINUX 设备驱动程序(第三版). 北京：中国电力出版社，2006：9~14

[42] 刘淼.嵌入式系统接口设计与 Linux 驱动程序开发[M].北京：北京航空航天大学出版社，2006：8~13

[43] 周敬利等. Linux 平台多通道 MPEG-4 视频捕获卡驱动程序的设计与实现[J].计算机工程与科学，2004，26(9)：25~26，74

[44] 魏洪兴，胡亮，曲学楼. 嵌入式系统设计与实例开发实验教材 II——基于 ARM9 位处理器与 Linux 操作系统. 北京：清华大学出版社，2005，12：18-19

[45] Introduction to Intrinsics.http://softwarecommunity.intel.com/articles/eng/3369.htm

[46] ZB.Chen，P.Zhou，Y.He. Fast Motion Estimation for JVT[A]. JVT-G016.doc，Joint Video Team(JVT) of ISO/IEC MPEG&ITU-T VCEG，7th Meeting[C]：Pattaya II， Thailand， 2003：7~14

[47] 孙宏; 薛骏; 凌青;Cache 中 TLB 的设计及优化

[48] H.Schulzrinne,A.Rao,R.Lanphier.Real Time Streaming Protocol.April,2000

[49] 孙学康，石方文，刘勇.多媒体通信技术[M].北京:北京邮电大学出版社，2006

[50] 樊姗，基于 RTP 的 H.264 视频传输技术的研究 硕士学位论文 山东大学 2008

[51] 史凯，基于 linux 系统的 H.264 视频监控系统的实现 山西电子技术 2008

[52] Iain E.G.Richardson.H.264 and MPEG 一 4 Video ComPression:Video Coding for Next 一 generation Multimedia[M].John Wiley & sons，2003

致　谢

江苏大学两年半的研究生学习和生活，是我永远无法忘记的一段经历。在这里，我得到了许多收获。

首先，我要由衷地感谢我的导师陈祖爵老师。在求学和科研期间，陈老师在我的专业学习、课题开展和论文撰写等方面都做了悉心的指导。他对我们学习科研的严格要求，使我们在专业上学有所成；他的睿智、敬业精神和责任感给我树立了工作学习的榜样；此外他还给我提供许多独立工作、实践的机会。在此，我向陈老师表示真挚的谢意！

其次，我要感谢闫述老师一直以来对我的关心和指导，正是她慈母般的关怀，才让我能够放下一切包袱好好前行！

再其次我要感谢朱轶老师在学术上给我的指导和帮助！

另外感谢在课题上帮助过我的人，他们是：冯志伟、付宪祥、麻勰光、欧阳烨龙、钟智慧以及师弟师妹们。谢谢他们给予我的帮助。

最后，我还要感谢我的家人。在我攻读硕士学位期间，他们在各方面给了我许多支持，正是由于他们的关心和鼓励，才让我能全身心地投入到学习和研究中。

最后，衷心感谢所有关心、帮助过我的人！

攻读硕士期间的科研情况

发表的论文

[1] 第一作者. "Efficient Fast Motion Estimation Optimization Based on the Improved Directional Search with mode detection for H.264." IEEE ICMCI 2010 EI 及 ISTP 检索

[2] 第二作者. "A Fast Motion Estimation Optimization for H.264/AVC encoder." IEEE WCNIS2010 EI 及 ISTP 检索

[3] 第二作者. "Efficient Fast Motion Estimation Optimization Based on the Improved Cross Search Template for H.264." IEEE ICISE 2010 EI 及 ISTP 检索

参与的科研项目

[1] 镇江市内河交通突发事件应急数字化智能指挥系统

[2] 电子书 Ebook(BSP 移植)